(2)

PROCESS DEFINITION AND PROCESS MODELING METHODS

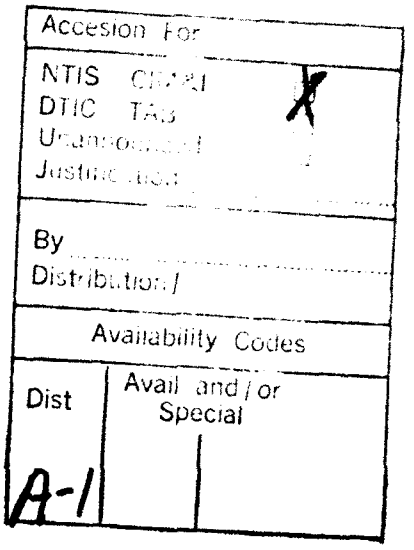
MDA 972-92-J-1018

SPC-91084-N

VERSION 01.01.00

SEPTEMBER 1991

PROCESS DEFINITION AND PROCESS MODELING METHODS



SPC-91084-N

Statement A per telecon Jack Kramer
DARPA/SISTO
Arlington, VA 22203

NWW 3/24/93

VERSION 01.01.00

SEPTEMBER 1991

Robert Chi Tau Lai

Reprinted for the
VIRGINIA CENTER OF EXCELLENCE
FOR SOFTWARE REUSE AND TECHNOLOGY TRANSFER

February 1993

SOFTWARE PRODUCTIVITY CONSORTIUM, INC.
SPC Building
2214 Rock Hill Road
Herndon, Virginia 22070

Software BackPlane is a trademark of Atherton Technology.

UNIX is a registered trademark of AT&T.

CONTENTS

FIGURES

TABLES

This page intentionally left blank.

ACKNOWLEDGEMENTS

Portions of Section 2 were based on the paper, "A Formalization of a Design Process," written by Jim Kirby, Robert Lai, and Dave Weiss for the 1990 Pacific Northwest Software Quality Conference. Special thanks go to Jerry Doland, Mary Eward, Shawna Gregory, Jim Kirby, and Dick Werling for reviewing the document.

This page intentionally left blank.

EXECUTIVE SUMMARY

The Software Productivity Consortium is working in the area of process improvement to help member companies better manage large, complex software development projects. This report is part of that process improvement effort.

To put this report in perspective relative to some of the other projects being conducted at the Consortium, consider some of the ways in which the Consortium is already helping member companies improve their process:

- The Consortium offers the Software Engineering Institute (SEI) Software Process Assessment for its member companies. By formally examining a company's current process practices, it is able to offer constructive criticism regarding ways in which to improve the process of developing software.

- The Consortium offers a variety of guidebooks that help guide the practicing engineer to perform his task more efficiently. Examples include the *Software Measurement Guidebook*, which focuses on metrics collection and analysis, and the *Evolutionary Spiral Process Guidebook*, which specifies a process incorporating risk management and iterative development and specifies methods to implement the process.

- The Consortium prototypes a software engineering environment that will provide process management tools for supporting automated software development.

This report forms the foundation of the bridge between guidance and automation. It offers a process notation that will enable a process engineer to model any process to any level of detail. It is this flexibility that makes this process notation so powerful and the precision that is conducive to process automation.

By building on the simple paradigm that people (roles) and resources perform activities that lead to products (artifacts), and by providing a rich set of relationships among those basic ingredients of a process, it is possible to model virtually any process and to maintain state tables that maintain information about the completeness of the product. Therefore, it is possible to know, at any given time, exactly what the state of completeness of all artifacts is, and hence the state of completeness of the activities and project. This provides the project manager with great insight and control over the events in the project.

Furthermore, by offering a notation that is flexible enough to allow state transitions based on the partial completion of activities, it is possible to model both deterministic and nondeterministic events. Hence, by allowing a partial ordering of events, it is possible to take advantage of opportunities or address risks as they present themselves.

The *process definition* techniques described in this technical report can be used by member companies to:

- Communicate with other software system developers by describing their process in a common, shared language. The implementation of this language will enable the generation of a software-process-centered *project management* environment for a specific software *project* development.

- Better understand the processes they are practicing.

- Describe their software process precisely so that they can plan their software development process and argue and reason their software process among their co-workers to get agreement.

- Plan process improvements for the future.

The process modeling notation described in this report provides benefits to member companies because it:

- Provides a precise, unambiguous description of the process.

- Provides a basis for building and integrating process tools, thereby recording process measurements, enforcing standard representation, and automating parts of the process.

- Allows all parties concerned (technologists, developers, managers) to agree (standardize) on the process.

- Provides a basis for process improvement/*process evolution*.

- Provides information for process and project management to reason about the status of a project, thereby providing insight as to how the process may be made more efficient.

1. INTRODUCTION

It is useful to try to formalize the description of a software *process* to understand it better, provide for automated support, provide *guidance* to software developers, and improve the software process. The Consortium is undertaking the software process improvement *task* for its member companies by:

- Conducting SEI self-assessments for member companies.
- Offering member companies the *Evolutionary Spiral Process Guidebook* and *Software Measurement Guidebook* for practice.
- Prototyping environment-based process management tools for supporting automated software development.

Member companies can use the *process definition* techniques described in this technical report to:

- Communicate with other software system developers by describing their process in a common, shared language. The implementation of this language will enable the generation of a software-process-centered *project management* environment for a specific software project development.
- Better understand the processes they are practicing.
- Describe their software process precisely so that they can plan their software development process and argue and reason their software process among their co-workers to get agreement.
- Plan process improvements for the future.

The process modeling notation described in this report provides benefits to member companies because it:

- Provides a precise, unambiguous description of the process.
- Provides a basis for building and integrating process tools, thereby recording process measurements, enforcing standard representation, and automating parts of the process.
- Allows all parties concerned (technologists, developers, managers) to agree (standardize) on the process.
- Provides a basis for process improvement/*process evolution*.
- Provides information for process and project management to reason about the status of a *project*, thereby providing insight as to how the process may be made more efficient.

1.1 OVERVIEW AND ORGANIZATION

This document describes how to define and model a software process through the use of graphics, forms, and a detailed language that enable automation of the process in an environment designed for that purpose. The organization and content of this document is as follows:

- Section 1, Introduction, provides the reasons why a more formalized process is needed. It provides an overview of the software engineering process and describes the elements of a software process.
- Section 2, Process Definition, shows how a process can be thought of in terms of *roles*, activities, and artifacts. This section provides the foundation for a way of precisely defining a process in terms of these basic components and the relationships among the components.
- Section 3, The Process Notation, provides three mechanisms for defining the generic process presented in Section 2: forms, textual language, and graphical diagrams.
- Section 4, Process Modeling Methods, provides a general description how to model and instantiate a process.
- Section 5, Implementation of a Process Notation, describes how to build a software engineering environment that enables an automated process.
- Appendix A, Process Definition Form Template Set, contains a set of blank forms that can be used to describe process rates, activities, artifacts, and relationships.
- Appendix B, A Formal Model of a Car-Driving Process Class, provides an example of how to use the process notation by illustrating the car-driving process.

Readers may refer to technical terms defined in the glossary in the back of this report.

1.2 INTENDED AUDIENCE

The primary audience for this report are member company technologists, that is, engineers who produce or evaluate products, services, and advanced technology and concepts in support of line engineers and *project managers*. These engineers evaluate a newly developed process or method and may recommend its use on a trial basis without impacting existing techniques or project schedule.

1.3 TYPOGRAPHIC CONVENTIONS

This report uses the following typographic conventions:

Serif font	General presentation of information.
Italicized serif font	Words, expressions, abbreviations, and acronyms found in the Glossary and publication titles.
Boldfaced serif font	Section headings and emphasis.
`Typewriter` font	Syntax of code or software responses.

1.4 APPROACH AND RESULTS

This report describes a process modeling notation for a model and how to use it to model both a software process and a class of processes. The notation is based on a generic *process model*, a two-level state model. Section 2.3 gives a detailed description of this generic process model. This generic process model is a state machine model. It permits progress in software development to be characterized as transitions among states. The model has two levels. The lower level is based on the states of the artifacts produced during the software process; such states are called artifact states (A-states). In the upper level, states are defined in terms of artifact states and are augmented with descriptions of activities and the roles of people who may perform activities. Activities include sequences of operations and analyses that can be performed on artifacts within the state. The augmented states in the upper level are called *process states* (P-states). This report contains:

- The definition of process notation based on this model in graphical diagram form, table form, and textual language. This process notation forms the basis of the methods for modeling a software process.
- The methods/processes of designing a process as well as a class of processes.
- The possible implementation of the process notation.
- Some examples of process definition in these process notations.

Readers should be able to model their own software processes after reading this report.

The technologist can use the notation to model either:

- A class of processes (e.g., waterfall processes, spiral processes) in terms of representing a set of artifacts and A-state and P-state classes in this notation.
- A process for a particular software project:
  - By using a predefined class of processes, the language contains an object-based construct for instantiating classes of artifacts, classes of A-states, classes of P-states, and the like.
  - By using the notation to model a process.

Member companies may use this notation to model a class of processes which encapsulate the different possible processes to represent the company's *policy* or procedure. Then, technologists may model a class of processes described above. One example of this is the Consortium's attempt to model the classes of processes for the evolutionary spiral processes. The second kind of usage of this notation (described in Section 2) is for line managers of member companies.

1.5 AN OVERVIEW OF THE PROCESS DEFINITION AND PROCESS MODELING METHODS

According to the National Institute of Standards and Technology's (NIST) process definitions, a process is defined as:

- A series of actions intended to reach a goal, possibly resulting in products.
- A series of actions or operations conducing to an end.

Software process is not well understood in general. One of the most effective and most adopted approaches in computer science is to **divide and conquer**. This approach to process definition defines the process by defining its elements. If the whole process cannot be defined, a list of elements can be used to divide the process and try to describe either:

- The subpart.
- The partial effect of the process.
- Only the input and output of the process.
- Only the sequence of the subprocess.
- Only the events that occur in the process.

As the process modeler's experience of describing a software process grows, his understanding of the software process grows. Process modelers use the techniques of generalization, meta-process, macro, parameterization, encapsulation, object-oriented, procedure calls, abstraction, and instantiation to describe a software process. Just as software is a description based on an abstracted model of a real-world problem, software process description is based on an abstracted model of a software development process. From machine code to fourth generation language, several layers of languages represent different layers of abstraction. Compilers are developed to convey the real-world problem to lower level and machine-oriented abstractions. The trend is to use higher level abstractions which are close to the application for the process modelers' convenience. Different existing process modeling notations/languages show the application of experience from computer language development. The more high-level software process abstraction a process notation offers, the easier a software process modeler's job will be.

1.5.1 Elements of a Software Process

This section describes the process elements. These elements—activity, sequence, process model, resources, control, policy, and organization—are the basic abstraction of the process description in this report. Some of the definitions of elements are adopted from the Integrated Software Engineering Environment (ISEE) process definitions drafted by Peter Feiler and Watts Humphrey (Feiler and Humphrey 1991).

1.5.1.1 Activity

A software process is a sequence of events, tasks, or activities. One way to define a process is to identify a list of things that will happen in a process. This is the approach of the IEEE in its 1074 Software Life Cycle Standard. An activity is used to partition or group those processes without regard for the resources needed, the sequencing, and how to do it. The activity can also be related to:

- What should happen before or after this activity.
- What kinds of resources may be needed for this activity.

- How the trigger or sequence can be described.
- What the general rules are that apply across activities.
- How to describe the experience, heuristics, and algorithms.
- How to describe how activities relate to a project team.

An activity is the basic building block for a process description. For all of the information related to the information listed above, the process modeler must find a suitable set of activities to decompose the information so that the description of the activity with associated information will become the description of the process.

1.5.1.2 Sequence

A sequence represents the order of activities in a process. It can be described in the following ways:

- Direct sequence: Activity A can happen only after activity B already happened (e.g., CPM and PERT).
- A sequence based on the transition or transformation of inputs to outputs.
- A sequence defined by the order in which procedures are called.
- Programming constructs (e.g., loop, if-then-else, for, while, COND).
- A sequence based on the satisfaction of conditions before and after the same event takes place.

None of these definitions describe different kinds of software processes, nor do they support the software development, management, and technology development. The process notation described in this report offers a combination of the definitions as a solution, as well as the translation of the process notation between forms, external representation, and graphical representation to provide for more complete support for software process.

1.5.1.3 Process Model

A model captures a view of interest about a system in terms of a set of predefined elements. A process model represents the aspects of interest of a process or a set of processes. The technologist can analyze and validate a possibly partial process definition for the purpose of modeling certain characteristics of an actual process and, if enactable, simulates the modeled process. Process models may model at the *process architecture*, design, or definition level. At times, technologists use process models to predict process behavior. Process models themselves have an architecture, a design, and a definition.

1.5.1.4 Process Resource

Process resources are necessary for a software process, e.g., a *process agent*, equipment, time, and office space. From the process description point of view, a notation should allow the process modeler to describe:

- What it is by name.
- The unit which makes sense for the process modeler.
- How much is needed for which kind of activity.

A process model only contains the resources of interest about the process which is being modeled.

1.5.1.5 Process Control

Process control can be defined as the external influence over process enactment by other *enacting processes*. This influence may be driven by *process evaluation*, and it may be through control of the process enactment state through reassignment of resources or through a change of process goals through process evolution. Overall, the controls in software processes are either manual or automatic. A manual control comes through a human decision. An automatic control comes through the preplanned mechanism which the process engineer codes in notation.

1.5.1.6 Policy

A policy is a guiding principle. It is a *process constraint*, usually at a high level, that focuses on certain aspects of a process and influences the enactment of that process. For example, a company may have a policy for its software development process which represents the process model that the company would like all projects to follow. The policy should be detailed enough for different projects to be managed under the same framework and at the same time be flexible enough for project managers to tailor to their management style, project-specific need, and project group's existing conventions.

1.5.1.7 Organization

An organization is the hierarchical structure of process agents in terms of the physical grouping that corresponds to the logical grouping in terms of the role these agents play in a project. The organization should be flexible enough to tolerate project changes. The logical organization, in terms of roles, should be well thought out and robust enough to tolerate changes in the physical organization as required. The mapping from the logical organization to the physical organization is through a role assignment activity which assigns a process agent to a particular role.

1.6 CHARACTERISTICS OF PROCESS DESCRIPTION

A process description can offer different characteristics of functionality. These characteristics are used to describe a process notation and will give the reader the background and sensitivity needed to apply the process notation.

1.6.1 Levels of Abstractions

A process description may offer a certain degree of support for how general a process can be described. There can be more different levels of abstraction between mathematics and the real software process, but only four levels are of interest.

- Process: Represents a real software process for a project, e.g., implementing module 23 or performing a code inspection on module 23.

- Process model: Represents a set of processes where the process model catches the variation of the set of processes, e.g., spiral process model, waterfall process model.
- Generic process model: Represents a set of process models where each generic process model can be used to represent a process model, e.g., Entry-Task-Validation-Exit (ETVX) diagram, Petri Net, two-level state machine model, state machine, and Structured Analysis and Design Technique (SADT).
- Mathematics: Represents all of the possible generic process models, e.g., set, bag, group theory, graph, logic.

1.6.1.1 Representative Power

The representative power of a process description can be in terms of the following factors. These factors are not mutually exclusive; they overlap for the representative power they cover.

- Granularity: The degree of detail into which a process description can go.
- Sensitivity: The degree of efficiency for process control and management that can be applied on the process described.
- Automation: The amount of automation that can be applied on the described process.
- Formality: The coverage of formalisms over different descriptions of process elements.
- Practical: The sum of the degree of ease to use, ease to understand, ease to implement, and ability to describe the existing process.
- Scale: The coverage for different sizes of software projects.
- *Accuracy*: The degree to which the process description matches the intended process.
- *Precision*: The degree to which the process description completely specifies all the actions needed to produce accurate results.
- *Redundancy*: The degree of overlapping information about a process in one process description.
- *Robustness*: The degree to which the process rejects unauthorized process control and/or modification (intrusion).

1.6.1.2 Opportunistic Versus Procedural Behavior

A process description can support both the procedural and opportunistic behavior of a software process. A procedural process forces the process agent to follow the process description step-by-step, just as a computer program executes statement-by-statement. An opportunistic process allows the process agent to follow a process description with more freedom, being constrained only by the requirement that, before beginning an activity, the necessary preconditions exist. In this way, a line engineer can take advantage of opportunities that present themselves during development.

1.6.1.3 Encapsulation of Activity

A process can provide the ability to represent a process fragment, or activity, as a self-contained unit similar to a procedure in an Ada package. The line engineer can treat description as reusable assets, that is, he can execute reevokable process fragments with other process fragments to form a new process. Just as an abstract database is one way to encapsulate a data type, a process description can encapsulate an activity.

1.6.1.4 Modularity

A module of a process can be in terms of a set of activities or a working stage for software development. The process description can provide the ability to group a set of activities to form an activity or to decompose an activity into a set of subactivities. The process modeler needs to provide the description of interfaces between this activity and its subactivities and the interface between two activities to support the modularity of a process description.

1.6.1.5 Meta-Process

The process description can provide the ability to represent the variation of a set of processes in which the commonalty is considerable. The process modeler describes the process through the parameterization of a template or metaprocess. As soon as he fills in the value of the meta process, he will specify or describe the specified process of that set. This kind of metaprocess description differs from the modularity representation of a process. Modularity pays more attention to creating the hierarchical representation of a process, encapsulation pays more attention to the reuse of a process, and metaprocess pays more attention to the variation and commonality of a set of processes. The same construct in a process description language, the process notation, can provide them.

1.6.1.6 Connection to the Software Development Environment

One of the very important capabilities of process notation is describing the interface between the process and its enacting, computer-aided software engineering (CASE) environment. The description of the interface needs to be independent of the CASE environment or CASE framework so that a change in the environment does not impact the process description, and line engineers can use the reusable process asset in multiple software projects under different environments.

2. PROCESS DEFINITION

2.1 BENEFITS OF THE METHOD OF MODELING THE SOFTWARE PROCESS

There are several benefits to the Consortium's method for developing formal models of software design processes. First, the line engineer can apply the method to any development method or process that has defined artifacts and defined composition and dependency relations among the artifacts. Defined artifacts and relations allow the formalization of artifact states upon which the formalization of the process states and the process depend, and the formal models produced help in understanding the process, managing it, and applying it in developing software. Defined artifacts allow specification of the artifacts that record the software development. Defined artifacts and defined relations among them allow the specification of how to determine the extent of completeness of the software product. For artifacts whose completeness cannot be formally specified, the line engineer can use technical reviews and tracking of the issues that result from them to specify completeness.

Application of this method results in process models that are relatively simple and clear, describe realistic, opportunistic processes and traditional waterfall processes, and provide useful guidance to the line engineers applying the process. These characteristics make the models useful to those trying to understand, manage, and follow the process. The models tend to be simple because a two-level approach allows for the separation of concerns. For example, the line engineer can use the model to determine the state of the software development without being concerned with how to determine the state.

The process modeler can model a waterfall process by developing an ordering of the process states and by specifying in the precondition of each P-state that it cannot be entered until previous P-states have been exited. The spiral process is modeled in terms of classes of P-states which represent canonical quadrants and canonical rounds. Process modelers provide line engineers with guidance on what to do next based on the state of the development. Modelers could usefully provide such guidance either as part of the automated working environment, as automatic, on-line help, or as an appropriately indexed hard copy.

2.2 STRUCTURE OF A PROCESS DEFINITION

This section describes the relation of a set of key terms used for a process definition. Figure 1 shows all of the key terms with directed arrow connecting them. The structure is described by describing the relations among the key terms.

In the following list, each term represents one relation between two artifacts. Section 3 describes additional relations. The Glossary lists some of the terms which have specific definitions in this report, and Section 2.3 provides a more detailed description.

- ***Refer-to.*** Analysis on the software process always refers to an artifact, A-state, and P-state.

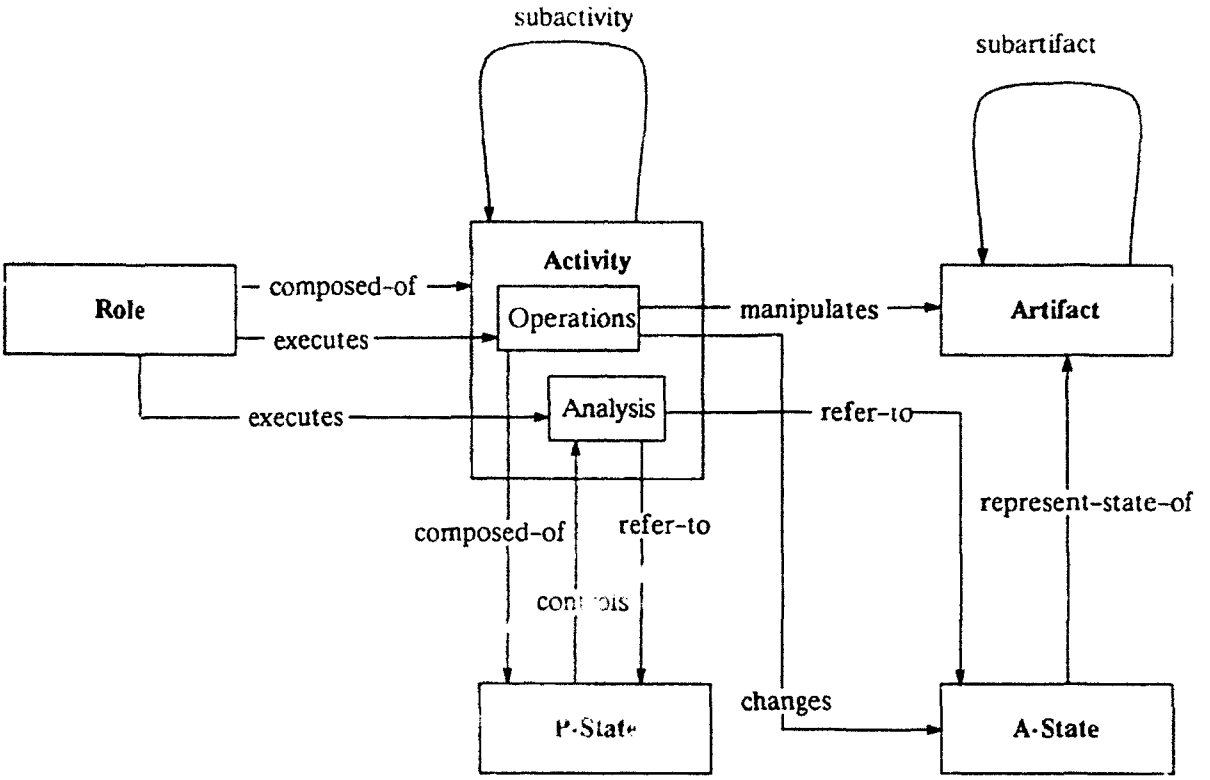


Figure 1. Relationships Defining the Design Model

- ***Composed of.*** A P-state is composed of the operations that the technologist can currently perform and a role is composed of the activities that the role is able to perform.
- ***Change.*** An operation changes (promotes or demotes) an A-state.
- ***Manipulate.*** Operations manipulate artifacts.

2.3 PRIMARY ELEMENTS OF A GENERIC MODEL

A software development process is a sequence of decision-making activities. Software development is a process of making decisions about what programs should implement the software requirements and what the required properties of those programs are, including properties such as their structure and interfaces.

Artifacts capture the decisions made during the software development process. Examples of artifacts are a description of the decomposition of a design into a set of components (such as modules, objects, packages, or subroutines) and a specification of the interface of one of the components. To characterize the state of a software development process, the line engineer must characterize the state of the artifacts produced during the software process, e.g., whether or not the software decomposition is complete. However, merely characterizing the state of the artifacts is insufficient to describe a complete software process. The process modeler must also describe the activities that may be performed on artifacts, the conditions under which those activities are performed, and the roles of the people who may perform them. For example, activities that the line engineer might perform on an interface specification of a software module are creating the specification, checking it for completeness and

consistency, and conducting a formal review of it. He performs the review after he has created and analyzed the specification for completeness and consistency. The reviewers may include designers other than the creator, implementors, and quality assurance personnel.

Software processes vary in many ways, including the way the software is organized into parts and the relations used to define dependencies among the parts. A particular software process may be defined as a sequence of decisions made in different states. The process modeler may model a methodology by a set of predefined states, i.e., it is a prescription for the artifacts to be used, the activities to be performed and their sequencing, and the roles that people play. The definitions of a set of states are the interpretation of the methodology. A line engineer using the methodology proceeds by following the activities prescribed by the states to produce the prescribed artifacts in the prescribed order. Figure 2, derived from Figure 1, illustrates the elements from which a generic software process model is composed; it is derived from (Potts 1989). Arrows in the figure indicate a relation between the elements that they connect, e.g., a role is composed of activities, an activity has subactivities, and an artifact has subartifacts.

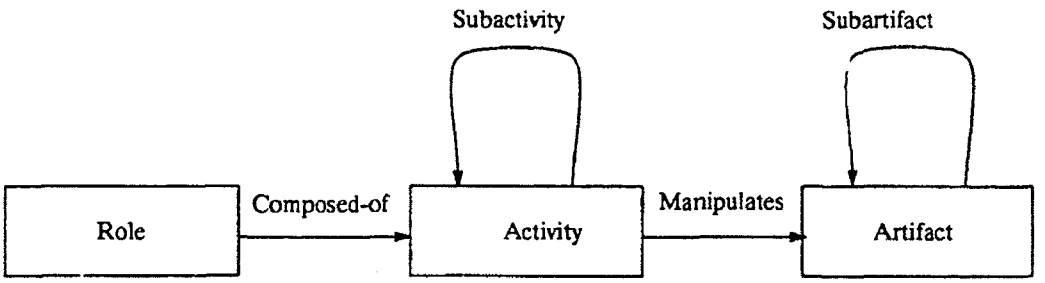


Figure 2. Elements of a Design Model

Artifacts are the work products of any software process and are typically documents. Most artifacts are composites or aggregates, i.e., they are composed of a number of parts, each of which is also an artifact. At some level of organization, the parts may be considered elementary, i.e., for the purposes of software development, they are not further decomposable or the decomposition is not of any interest to the process modelers.

The goal in identifying artifacts as part of a method is to find artifacts whose states the line engineers can easily assess. The line engineer needs to associate a rationale with an artifact. Whereas the artifact represents decisions that he has made, the **rationale** for the decision—what issues led to making the decision, what alternatives were considered, and what the justification was for the alternative chosen—is a separate concern from the decision recorded by the artifact.

The state model has two levels. The lower level is based on the states of the artifacts produced during the software process; such states are called artifact states (A-states). Because A-states alone are insufficient to describe the software process completely, descriptions of activities, operations on artifacts, analyses that the line engineer can perform on artifacts within the state, and the roles of the people involved augment them. The augmented states in the upper level state model are called process states (P-states). A complete description of the state model is presented Section 3.

The two-level model allows the separation of the process description from the representation used for the artifacts. For example, the upper level model may specify operations to be performed on interfaces to information hiding modules (Parnas 1972) without specifying the representation of such

interfaces. The specification of the representation may be confined to the lower level model, where the state of the artifact that represents such a module is defined. Figure 3, derived from Figure 1, shows the relationships between P-states, A-states, and the other elements of the process model. The remainder of this section defines the relationships referenced in Figure 3.

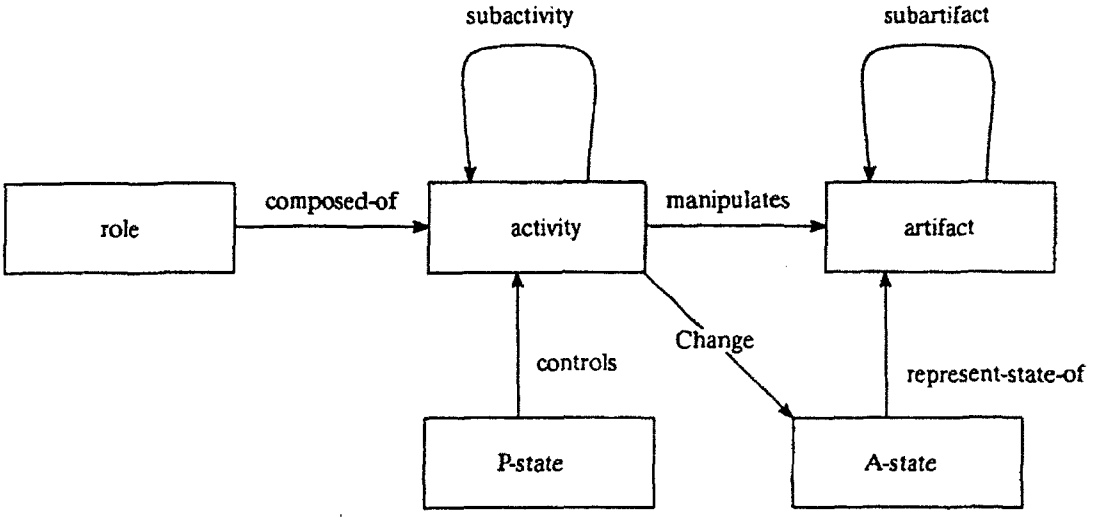


Figure 3. Relationships Defining the Design Model

Different software projects may choose to use different methods for any step or subprocess for their software processes. For different methods, the artifacts used are different. An activity is the work that goes into producing or manipulating an artifact, i.e., each activity is associated with a particular artifact. Activities whose artifacts are composite have subactivities, each associated with a component artifact.

Whether an activity is performable or not depends on the state of the artifacts. At any point in time, the set of performable activities represent the choice of artifacts on which the line engineer may work. Those that are not performable represent the artifacts on which he may not work. The model prescribes a permissive ordering of activities (and on the work of the developer) by specifying which activities are performable and which are not at any point. Permissive means that the line engineer is free to choose, from among the set of performable activities, those to pursue. The process modeler specifies a permissive ordering to support a realistic process. The model captures the information needed to produce a rational description of the work products of an opportunistic process (Guindon 1988).

In addition to specifying artifacts and activities, the process modeler also specifies the roles played by people involved in design. As with artifacts and activities, different processes will specify different roles, such as designer, reviewer, implementor, and manager. The logical grouping of activities in which people may participate define the roles.

By augmenting state descriptions with activities and other process-related information, the process modeler can analyze the needs of the line engineer at any point in the process. The goal of the analysis is to present to the line engineer the information that he needs at the time he needs it. It also helps to focus on the concerns of what guidance to give to the line engineer and what analyses the process modeler can perform on the software at any time. Finally, it helps to separate the concern of what information the process modeler should present to the line engineer from how the process modeler should present information to him.

2.3.1 The Artifact State Model

This section describes the A-state model, beginning with a description of the relation, structure, and rationale behind artifacts, followed by a definition of the A-states used to specify the states of artifacts, an elementary artifact (i.e., one that contains no other artifacts), and a composite artifact (i.e., one that is composed of other artifacts).

2.3.1.1 The Relation and Structure Among Artifacts

Artifacts are the components that result from the decomposition of any software-development-related object. Relations among a group of artifacts create the structure of artifacts. Two predefined relations among artifacts are depends-on and is-composed-of. The relations described here are not necessarily a physical relationship among different artifacts. The relations are treated as an extra marking for process modelers to create relationships whenever necessary. The implementation of a language may implement the predefined relation is-composed-of as a physical object decomposition relation. It is up to the implementation. The two predefined relations create two different kinds of structures among artifacts. The depends-on relation records the dependency structure which describes decisions about what artifacts are affected by changes to other artifacts. If, for artifacts X and Y, the convention (X, Y) is used to say that X depends on Y, X can be present and correct if and only if Y is also present and correct. X, then, is affected by changes to Y.

The other possible structure is an aggregation of modules and the is-composed-of relation. The is-composed-of relation, a relation among the artifacts, defines a tree. An artifact in the tree is composed of its children. The artifacts at the leaves of the tree are work assignments for individual software professionals or for a team of software professionals. Besides work assignments, the process modeler can define artifacts from different kinds of needs of software processes. The artifacts can be abstract artifacts or physical artifacts. The physical artifact is the artifact which really contains data objects related to software products. The abstract artifact is the artifact that the process modeler creates for the process model's sake.

2.3.1.2 The Rationale for an Artifact

By default, each artifact has a rationale associated with it. Since this is by default and is general for every single artifact, both predefined and user-defined, it is not part of the process notation. One of the attributes of a rationale is open_issue. Because a software process is a sequence of decision-making activities and the artifacts are the work products of any software process, the process model must be able to capture the rationale for the entire software process. The predefined open_issues will return a boolean value to reflect if there are open issues associated with the particular artifact.

2.3.1.3 The Artifact States

English definition list and informally define the A-states. The A-state of an artifact describes the completeness of the artifact. The process modeler defines a partial ordering relation on the A-states which reflects the permissive ordering of activities. The A-states can be thought of as defining a state machine for each artifact. Activities performed by software professionals and other participants in the software process cause the state machines for the artifacts to transition from one A-state to another. The line engineer uses the A-states, a formalization of the completeness of the artifacts, as a guide in completing the software artifacts.

Because the process modeler has defined a partial ordering on the A-states, the line engineer can use the relational operators equal, greater than, less than, greater than or equal, and less than or equal on A-states. For example, for A-states M and N:

$$N > M$$

means that an artifact in A-state N is more complete than an artifact in A-state M. The three default A-states which come with all of the artifacts are:

- ***Empty.*** There is no work associated with the artifact. It is empty. By default, it is the very first A-state at which an artifact will be.
- ***Named.*** The artifact is given a name. Giving an artifact a name is also a decision. Therefore, there is one A-state to document the activity of naming an artifact. Implementation requires a system identification for all existing artifacts.
- ***Completed.*** No more work needs to be done for this artifact. By default, it is the very last A-state at which an artifact will be.

2.3.1.4 Elementary Artifacts, Composite Artifacts, and Predicates

The line engineer determines the state of an elementary artifact, one that is not an aggregation, by examining the artifact itself, whether there are any open issues that annotate it, and the state of artifacts on which the artifact depends. The line engineer determines the A-state of an aggregate artifact by the state of its components and the A-states of artifacts upon which the aggregate artifact and its components depend.

Table 1 lists a set of predicates that are available for a user-defined relation, user-defined artifacts and a user_defined object class. They are the only predicates which the line engineer can use to define the A-state of an artifact. Since he defines all of the definitions of an A-state on top of this set of predefined predicates, the completeness of implementation ior this process notation is possible. The Empty, Named and Completed A-states are from the default A-state. The State_of is a predicate for retrieving the A-state by name of the artifact, "x".

Table 1. Definitions of Predicates

Predicate	Definition
Empty(x)	There is no work associated with the artifact instance x. The artifact is empty.
Name(x)	There is a name associated with artifact x.
State_of(x)	The A-state of x, where x is an artifact.
Open_Issues (x)	There are open issues associated with object x.
Completed(x)	Artifact x is totally completed.

2.3.2 The Process State Model

Table 2 shows the components of a P-state. The A-states of the artifacts associated with the P-state characterize a P-state. Consequently, the specification of a P-state includes a list of the artifacts associated with it.

Table 2. Components of a Process State

Components	Definitions
Name	The name of the state.
Entrance Condition	Condition required for entry of an activity. The line engineer uses this to determine when a process can enter a particular P-state.
Artifacts	A list of artifacts upon which work may proceed in the P-state.
Information Artifacts	The artifact which holds information required to support operations and analyses in this state. They are either artifacts which are returned by the analysis function or artifacts analyzed by the analysis function. These artifacts may include raw data and statistical data calculated by the analysis function.
Activities	A list of operations and analyses that the line engineer may perform in the P-state. See Tables 3 and 4 for detailed descriptions of operations and analyses.
Post-Action	Action and predicates on A-states that determine when the process executes the action.
Exit Condition	Predicates on A-states that determine when the process exits the P-state.

In addition to artifacts associated with the P-state, a P-state specification also describes the entrance condition for a state entry and the activities that the line engineer may perform in the state. The entrance condition is the condition that must be true to enter the state. The activities that the line engineer may perform in the state are of two types:

- Operations, which the line engineer may use to change the state of an artifact.
- Analyses, which return information about the software development in general, including guidance on development methods, and artifacts in particular.

2.3.2.1 Operations

A precondition, the artifacts to which the operation may be applied, actions, roles, and post-condition, may specify an operation as shown in Table 3. Preconditions are represented as predicates on A-states that determine when the line engineer can perform the operation. Post-conditions are represented as predicates on A-states that represent the condition when the operation is done.

Table 3. Components of an Operation

Components	Definitions
Precondition	Predicates on A-states that determine when the line engineer can perform the operation.
Artifacts	The artifact manipulated by the operation.
Action	The function to be performed by the operation.
Roles	A list of roles of people who are allowed to perform this operation.
Post-Condition	Predicates on A-states that represent the condition when the operation is done.

The line engineer may perform an action when the precondition for the operation is true. The action embodies the effect of the operation. Accordingly, the description of an action is a formal specification of the effects of the operation. As in other examples, the actions are assumed to be primitive functions and thus have not been defined in the process model in detail. They can be a tool, a micro-step in the process, or whatever size of step in the process at which the process modeler wants to model it in detail. Since not all operations are permissible to all people who participate in design, an operation also has a list of roles of people allowed to perform the operation.

2.3.2.2 Analyses

In a P-state, a line engineer can get information about the current status of the software project by invoking analysis. The information offered by the analysis helps line engineers make decisions. Table 4 summarizes the components of analysis and their definitions.

Table 4. Components of Analysis

Components	Definitions
Artifact	A list of artifacts related to this analysis.
Analysis	Specification of the analysis to be performed.
Roles	List of roles of people who may perform the analysis.
Trigger	How the line engineer can invoke the analysis.
Result	The handling of the analysis results.
Action	Actions to be performed depending on the results of the analysis.

An analysis is defined by the artifacts to which it applies, the functions it performs, its types, and the roles of the people who may invoke it. The following attributes determine the type of each analysis:

- ***Trigger.*** The line engineer can obtain an analysis in a predefined temporal situation, e.g., when entering a P-state, exiting a P_state, whenever decided by the user, periodically, or before or after a certain operation.

- ***Result.*** The line engineer may capture the result of analysis in some persistent storage, external file, or predefined data object class and shown the result in a predefined diagram type or predefined table type in a predefined text report type.

- ***Action.*** The line engineer may specify response actions as part of the analysis definition for the cases when he finds certain conditions to be true from the analysis. Example actions include:

  - ***prompt message (A-state, message).*** This action prompts the line engineer(s) with the message when the analysis function returns A-state.

  - ***save (A-state, name).*** This action saves the result of analysis in a file called name when the analysis returns A-state.

  - ***mail reviewer (A-state, message, list of user_ID).*** This action mails the message to the line engineers (list of user_ID) when the analysis returns A-state.

2.3.2.3 The Process Model as a State Machine

Because line engineers are likely to be working on several aspects of a software project concurrently, the process modeler models the software process as a concurrent state machine, i.e., a state machine that can be in several states simultaneously. This permits a line engineer to make progress in one state for a while and then move to another without completing the first. It also permits several line engineers to work on different parts of the software concurrently. By offering this capability, the process model allows the development team the freedom to act opportunistically yet still provide a means for guiding them in an orderly way. In addition, they can use it to record an opportunistic process so that they can plan the process and still hold the flexibility for the line engineer to adjust to real situations when enacted.

Modeling the process as a concurrent state machine satisfies several concerns. First, it provides a precise definition of the methodology that the project needs. Designers, reviewers, and others involved in the process have a reference that specifies what they can do at any stage of the process. Second, it permits a parallel process since the model identifies opportunities for different line engineers in a team to work concurrently and independently. Third, it provides a specification for a tool to support the process, i.e., a tool that can keep track of currently active P-states, record information about the software development, and provide the operations and analyses necessary for software professionals to make progress on the software project.

2.4 ROLES

A role is a logical grouping of activities. A role defines the responsibilities of jobs associated with the software development task, such as designer, manager, or reviewer. A person who plays a designated role can determine what activities he may perform by looking at all the operations and analyses permitted for that role over all states. Roles are expected to vary according to the hosting organization as w as the method used, e.g., a reviewer's role in one method may be quite different than in another method.

This page intentionally left blank.

3. THE PROCESS NOTATION

For a software process to be fully described under the Consortium's generic process model, a notation is necessary for the process modeler to precisely define the process in his mind. The line engineer can use this same notation to represent the necessary process-centered software environment for a project. Different projects may have different processes according to their needs, and the process-centered working environment should reflect that. The process modeler can do mapping by interpreting the software process description in a formal process notation and generating the activities coordination program to integrate tools. So far there are three forms of notation:

- *Form.* The process notation is in terms of a set of forms, e.g., a form for describing an artifact, a form for describing a P-state. The process-modeling process is just like the form-filling process. Each form represents a template which asks for some data to be provided to describe the process.

- *Textual Language.* This is the most common practice for formal notation and languages in the world of computers. It is also the centerpiece of the Consortium's generic formal process notation. Together the form and graphical diagrams should contain a full set of information that the textual language will cover. A lex/yacc form of representation as well as Backus-Naur Form (BNF) representation of the language has been developed by the Consortium.

- *Graphical Diagram*: A graphical diagram notation is designed for the convenience of the process modeler. A process modeler should try to use the graphical diagram to model his process until he runs out of means.

It is suggested that, when reading the following sections, readers refer to Appendix B, which provides an example of the car-driving process, before reading the following sections. This will enhance understanding of the forms.

3.1 FORMS

The following sections explain how to fill in each of the forms, with explanation of what kinds of data need to be provided, what fields are related among the different forms, related forms, and how to interpret the information in the forms. They also explain the rationale for the default values which come with the implementation of the language. Some of the data for the form is formal. It is exactly the same as the data for the textual language. Some of the data is informal, i.e., an English description of an artifact. Appendix A provides a set of forms that the reader can tailor to his needs.

3.1.1 Artifact Definition Form Template

Table 5 shows the Artifact Definition Form used to record information about artifacts.

Table 5. Artifact Definition Form Template

Artifact Definition Form			
Name			
Synopsis	(English description here)		
Complexity Type	(Elementary or Composite)		
Data Type	(Name of object class, predefined or user-defined)		
A-State List			
A-State-Name	Put condition lists over A-state of dependent artifacts here, if artifact is composite artifact.	(English explanation here)	
Subartifact List			
First Subartifact Name	(Name)	(English explanation here)	
Relations List			
Name of relation	(English explanation here)		
	Type of relation (predefined or user-defined)	Direction (to or from)	Name of related artifacts

The form has the following fields:

- ***Name.*** Each artifact needs to have a name assigned to it by a process modeler. If the process modeler is modeling a class of artifacts, he should suffix the name with a dot (.). If he is modeling an artifact from an existing, predefined class of artifacts, then artifact-class-name.name will be the name. If he is modeling just an artifact, what he needs to put in is a name. This naming convention is the same for all of the names in terms of modeling a process or a class of processes. In other words, this notation adopts the naming convention of using a dot to represent the relation between class and instance.

- ***Synopsis.*** This is the English explanation of the artifact being defined. Every form in this process notation has a synopsis field to be filled in. An entity in this column is optional, but it is strongly suggested that they be filled to increase the understandability of the process model. This report does not explain how to fill in the synopsis for the rest of the forms in this process notation.

- ***Complexity Type.*** The process modeler can enter only one of the two reserved words, elementary or composite, in this column. A composite artifact is one that consists of subartifacts. An elementary artifact is indivisible.

- ***Data Type.*** If the complexity type of an artifact is composite, the artifact does not have a data type. In that case, it is optional for the process modeler to provide anything in the Data Type field. The process modeler does not and will not define the implementation of composition relationships among superartifact and subartifact. It is the implementation of the process-centered software environment that makes the selections. It can be under the real object class hierarchy in the repository of a software project or just implemented as referencing mechanisms which reflect the subartifact relationships. Only the name of the object class can be put here. In other words, all

of the names here need to end with a dot. When the process modeler or line engineer enacts the process model, the implementation generates the name of the object's instance and maintains the name for them. They can provide two kinds of names here:

- ***Predefined Object Class Name.*** So far, the language only supports two data types, integer and string or text, and three basic types of object classes:

  Generic formatted text, e.g., the catalog name of the Interleaf text-formatting template set.

  Generic table, e.g., the number of the row and column and the data type of cells.

  Generic stick-ball diagram, e.g., the data types associated with each kind of stick and ball, the possible connections from kinds of sticks to kinds of balls and vice versa, and the name of the predefined graphic symbols and line font for the diagrams.

  The process modeler needs to provide the name of the class with a list of (parameter name, value) pairs which denote the instantiation of the generic object class for this artifact.

- ***User-Defined Object Class Name.*** The process modeler needs to provide the name of the object class and the instantiation parameters/instantiation value pair list.

- ***A-state List.*** There are no dots associated with the name for the A-state in the A-State List field. The process modeler uses the condition lists to present the conditions when the artifact will be in this particular A-state. The condition representation described here is the same as in all of the other places in the process notation. The condition may be composed of the following kinds of predicates:

  - State_of(x): An A-state retrieval predicate that comes with all predefined and user-defined artifacts by default.

  - Empty(x): A default predicate that returns a boolean to indicate whether an operation has been applied to an artifact after it was created. This default predicate comes with any predefined and user-defined artifact.

  - Open-issues(x): A default predicate that comes with all of the artifacts. By default, the process modeler can associate all artifacts with an issue database. Open-issue returns a boolean, indicating whether the artifact has an open issue in the issue database.

  - User_defined_relation( ): A default predicate that comes with any predefined and user-defined relations.

  - There_is_one: There exists one or more artifacts.

  - For_all: Refers to all of the artifacts.

  Only the following connectors for the predicates are allowed:

  > or GT: Greater than.

  > = or GE: Greater than or equal to.

< or LT: Less than.

< = or LE: Less than or equal to.

= or EQ: Equal.

The process modeler can use parentheses to clarify the meaning of the conditions. He can use only there_is_one and for_all, and he is only allowed to apply them over sub_artifact( ) and user_defined_relation( ), two kinds of predicates.

If an artifact is an elementary artifact, the process modeler is not required to put in the condition expression for each A-state. If the artifact is a composite, the condition list is required.

- ***Subartifact List.*** The process modeler needs to list only the subartifact names in the Subartifact List field. An English explanation is optional.
- ***Relation List.*** The information captured in the Relation list reflects the necessary relations for the particular artifact that the process modeler wants to have. A name relation as either predefined or user-defined, the type of relation, the direction of relation, and the name of the class of artifact to which this artifact will be related. Again, the process modeler can provide an optional English explanation. See the example in Appendix B.1.1.1.

3.1.2 Relation Definition Form Template

Table 6 shows the template for the Relation Definition Form. This form represents the user-defined relations. The relation table example are in Appendix B.2.1 through B.2.4.

Table 6. Relation Definition Form Template

Relation Definition Form			
Name			
Synopsis	(English description here)		
Possible Artifact Pair List			
From		To	
From		To	
From		To	

For each relation, the process modeler needs to fill in the following fields:

- ***Name.*** This is the name of the relation.
- ***Synopsis.*** See Section 3.1.1.
- ***Possible Artifact Pair List.*** Each pair represents a directed relation from one class of artifact to the other.

3.1.3 Process State Definition Form Template

Table 7 shows the template for the Process State Definition Form. This is the form to use to define a P-state. The process state table examples are in Appendix B.4.1 through B.4.3.

Table 7. Process State Definition Form Template

Process State Definition Form		
Name		
Synopsis	(English description here)	
Main Role		
Entrance Condition		
Artifacts List		
Information Artifacts		
Operation List		
	Name	
	Description	
Analysis List		
	Name	
	Description	
Post-Action List		
	Description	
	Condition	
	Action Function	
Exit Condition		

The form has the following fields:

- *Name.* This is the name of the P-state.
- *Synopsis.* This field summarizes the purpose of the P-state. See Section 3.1.1.
- *Main Role.* This field optionally defines what role is most responsible for completing the activities associated with the P-state. The main role has the ability to reassign role play for the P-state and to invoke the default operation version, baseline, commit, version_query, and roll back to some earlier version of the artifact. The process modeler needs only to fill in the role name for the main role column. The main role also has the right to assign roles for invoking a new P-state operation.
- *Entrance Condition.* This field defines the condition required for entry into this P-state.
- *Artifacts List.* This is the list of artifacts operated on for this P-state. The components of this list may be in terms of artifacts or classes of artifacts:
  - A list of artifacts when modeling a process.
  - An artifact class list when modeling a class of processes.
- *Information Artifact.* This is a list of artifacts which represents all the artifacts that the activities, operations, and analyses reference. They can be in terms of artifacts or classes of artifacts:
  - A list of artifacts when modeling a process.

- An artifact class list when modeling a class of processes.
- An exit condition (the same as the entrance condition).

- ***Operation List:*** This is a list of operations for a P-state or class of P-states to model a class of P-states. The name of an operation may be the name of a class of operations if the operation is listed as an instantiation parameter of this P-state. The name of an operation is the name of an operation; this means that the operation is an invariant of this P-state. To model a P-state, the name of operation may be an instance of a class of operations or it may be the name of an operation only defined for the use within this P-state.
- ***Analysis List.*** This is the same as the name of the operation listed in the Operation List field.
- ***Post-Action List.*** Each post-action contains three parts:
  - Description: An English description of the post-action.
  - Condition: The condition that must be true before the process modeler or line engineer can invoke the action.
  - Action: The action to take place.
- ***Exit Condition.*** This field defines the condition required for exiting this P-state.

3.1.4 Operation Definition Form Template

Just as an artifact is defined on top of object classes, operations are defined on top of methods/services of object classes. Table 8 shows the template for the Operation Definition Form, which is used for this purpose.

Table 8. Operation Definition Form Template

Operation Definition Form		
Name		
Synopsis	(English description here)	
Precondition		
	Operation Type	(Manual, tool, function library, or state_control)
	Artifact List	
	Action	
	Role List	
Post-Condition		

The definition of an operation (class) should cover:

- ***Name.*** This is the name of the operation.
- ***Synopsis.*** This is a brief description of the operation, what is operated on, and the result. See Section 3.1.1.

- ***Precondition List.*** This is a list of conditions. It is represented in the same way as the conditions for entrance and exit conditions of a P-state.
  - ***Operation Type.*** There are several operation types that the process modeler can supply in this field:

    Manual. The operation is not automated or not modeled inside the process model being created; i.e., it is a black box to the process model. The person performing the role types in the password to signal that the task is done.

    Existing Tool. The action for the operation is a commercial off-the-shelf (COTS) tool.

    Function Library. The action for the operation is either a library routine in the programming library or a shell script.

    State Control. The operation is encapsulated by evoking a new P-state or by changing an A-state of a particular artifact, and the result of it opens the entrance condition of some P-state.
  - ***Artifact List.*** This is a list of artifacts upon which the operation is acting.
  - ***Action.*** This is the name of the COTS tools, shell scripts, or library routines.
  - ***Role List.*** This is a list of roles that may perform the operation.
  - ***Post Condition.*** This is the same as the precondition.

3.1.5 Analysis Definition Form Template

Similar to an operation, an analysis is defined on top of a method/service of object classes. Table 9 shows the template for the Analyses Definition Form.

Table 9. Analysis Definition Form Template

Analysis Definition Form		
Name		
Synopsis	(English description here)	
Information Artifacts		
Analysis Function		
	Analysis Type	(Management, quality, consistency, process status, or user-defined)
	Artifact	
	Analysis Function	
	Role List	
	Trigger Type	(State_entrance, state_exit, user, periodical, or operation)
	Result Type	(File, data_structure, diagram, table, text_report, or form)
	Action Type	(Prompt message, save to file, or mail to role)

The definition of an analysis (class) should cover:

- *Name.* This is the name of the analysis to be performed.
- *Synopsis.* This is a brief description of the analysis to be performed, with a general description of the output. See Section 3.1.1.
- *Information Artifacts.* This is the list of artifacts that this analysis may reference. This field is more an aid for implementation than it is used for process modeling. If the implementation is very strong, this column may be omitted.
- *Analysis Function.* This is the name of the method/service for the implementation.
  - *Analysis Type.* The kinds of analyses that can be performed include:

    Management. The analysis function generates information for management decisions.

    Quality. The analysis function generates information for the quality of the work product.

    Consistency. The analysis function generates information for the consistency of various work products.

    Process Status. The analysis function generates information for the currency of the process.

    User-Defined. The process modeler can define analysis function. If it is a tool, the analysis should be well-packaged as a stand alone tool. If it is a library routine, the analysis function needs to have a further integration step (e.g., compilation) to be incorporated into the process centered environment.
  - *Artifacts.* This is the list of artifacts that the implementation programs takes as input.
  - *Analysis Function.* This is the implementation of the analysis function.

    In addition to the standard, predefined analyses given above, the technology process modeler may define other analyses. These analyses may be autonomous and standalone, or they may be partial analyses that he can combine to perform a more sophisticated kind of analysis.
  - *Role List.* This is a list of roles who may start the analysis.
  - *Trigger Type.* Each analysis has a trigger type that determines when the process modeler or line engineer is to invoke the analysis. The different types are:

    State_entrance: The process modeler or line engineer invokes analysis each time the P-state is entered and before any activity is evoked.

    State_exit: The process modeler or line engineer invokes analysis automatically before exiting a P-state and after the post-action.

    User. The process modeler or line engineer can evoke analysis function anytime he wants it.

Periodical. The process modeler or line engineer invokes analysis automatically at a specified period of time, e.g., a day or a week.

Operation. The process modeler or line engineer invokes analysis before or after a certain operation is evoked.

- ***Result Type.*** This field describes the format of the output of the analyses. Some examples include:

  File. This is the name of a class of artifacts with an object type that represents a file format.

  Data_structure. This is the name of a class of objects.

  Diagram. This is a class name of objects that represents a class of diagrams.

  Table. This is a class name of objects that represents a class of tables.

  Text_report. This is a class name of objects that represents a class of formatted texts.

  Form. This is a class name of objects that represents a class of forms.

- ***Action Type.*** A single analysis may have more than one of the following:

  Prompt Message. There is a text string that represents a message.

  Save To File. This is a file name represented as an instance of a class of artifacts.

  Mail To Role. This is a list of roles and a text string or a file to be included in mail.

  The information needed in the form should be consistent, e.g., if the process modeler specifies file as a result type, then he should also specify save to file or mail to roles as the action in the Action Type field. The implementation needs to offer the debugging facility for it (see Section 3.2).

3.1.6 Action Definition Form Template

Table 10 shows the template for the Action Definition Form, which represents the user-defined action.

Table 10. Action Definition Form Template

Action Definition Form	
Name	
Synopsis	(English description here)
Type	(function_library, shell_script, tool)
Source Path Name	
Input Object Class	
Input Path Name	
Output Object Class	
Output Path Name	

The template includes the following fields:

- *Name.* This is the name of the action or operation to perform.
- *Synopsis.* This is a brief description of the action to be performed. See Section 3.1.1.
- *Type.* This field may include one of the following:
  - *Function_library.* The action is implemented as a routine in the programing library.
  - *Shell_script.* The action is implemented as an operating system command script, e.g., shell scripts in UNIX.
  - *Tool.* the action is a standalone tool.
- *Source Path Name.* The full path name represents where the action executable is stored.
- *Input Object Class.* This information is necessary for tools and shell scripts and essential for programming library routines.
- *Input Path Name.* This information is required for tools and optional for shell script and library routines.
- *Output Object Class.* This information is necessary for tools and shell scripts and essential for programming library routines.
- *Output Path Name.* This information is required for tools and optional for shell script and library routines.

3.1.7 Role Definition Form Template

Table 11 shows the template for the Role Definition Form, which is used to define the role for a process model.

Table 11. Role Definition Form Template

Role Definition Form	
Name	
Synopsis	(English description here)
Member List	
Activity List	

The template includes the following fields:

- *Name.* This is the name of the role, e.g., designer, reviewer, module designer, modeler/reviewer.
- *Synopsis.* This is a brief description of the role and the major activities it performs. See Section 3.1.1.

- *Member List.* This is a list of log-in names or a list of names for the position in an enterprise, e.g., manager_validation_lab or technical_lead_reuse_lib.

- *Activity List.* This is a list of activities which the specified role can activate.

3.2 TEXTUAL LANGUAGE

As explained in the introduction to this section, the process modeler does not use the textual language directly; rather, it is provided to enable the development of an environment that will provide automation support to the process. The Consortium has developed a candidate language. This language supports either the graphical representation of the process given in the following sections or the information that is collected and organized on forms as presented in Section 3.1.

Note that, given the many relationships among the various forms and components of those forms, a method of checking the consistency of data across forms is very desirable. To check the consistency without a tool would be time-consuming, tedious, and error-prone.

The process modeler will interface with the process notation through the graphics and the forms; the textual language is used for the environment implementation only. The process modeler generates the textual language representation of the process model on the back end of the form/diagram browser/editor for the process notation. He passes this textual representation to the compiler/translator/automatic instantiator to generate the process-centered software environment. It is up to the implementation to have the textual language or not. The compiler/translator/automatic instantiator can share the data structures/objects maintained in the form/diagram browser/editor, and the process modeler can generate the process-centered software environment directly.

3.3 GRAPHICAL DIAGRAMS

For convenience in representing relationships between roles, activities, artifacts, A-states, and P-states, the Consortium has designed a graphical diagram notation. Because of the complexity of the software process, however, it is not possible for the graphical notation to cover all of the semantics of the language and still maintain its simplicity for ease of process modeling. Therefore, the solution is to:

- Capture the main relationships of the subcomponents of the software process in a graphical diagram.

- Associate the secondary relationships in "property sheets" of the components.

- Develop the graphical diagram so that all of the information can be in one diagram if the line engineer wants it that way, and then the environment implementation can show the diagram in some simplified view.

The following sections present the elements of the graphical diagram and describe what they mean.

3.3.1 P-State and Operation

A P-state is represented as a rectangular box with two kinds, and only two kinds, of boxes that can intersect with it. Figure 4 shows an example. Figure 18 in Appendix B presents a P-state diagram for a process model of driving a car.

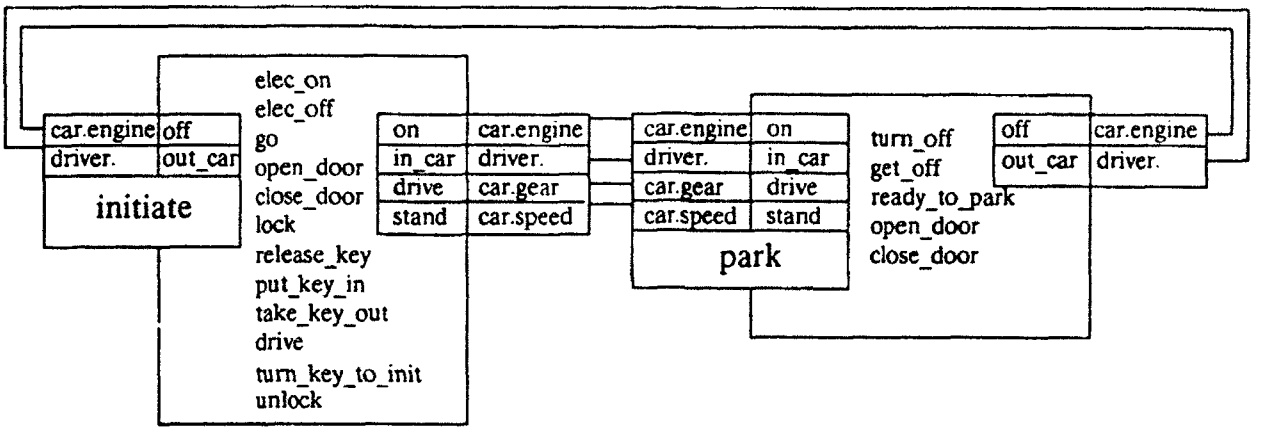


Figure 4. P-State Diagram Example

The first kind of boxes represent the entrance conditions, and the second kind represents exit conditions. The difference is that the entrance condition has a P-state name subbox on the bottom of the box, but the exit condition box does not. The P-state name can be on any edge of the P-state boxes, even in the same edges. This gives the process modeler a lot of geometric and topological freedom to draw. There are no arrows associated with the entrance conditions and exit conditions. The process modeler should identify the line directions very easily from the entrance or exit conditions boxes to which they are attached.

A set of cells divides the condition box. Each cell represents one artifact in one A-state which composes the entrance/exit conditions. The intersection lines between the condition box and the P-state box divides the cells into two parts. The parts outside of the P-state contain the name of the artifact, and the part of the cell within the P-state box represents the A-state of the artifact. The line only connects to the cell of the condition boxes.

Each artifact cell may have more than one line connected to it. This means the A-state change of an artifact may affect more than one P-state. More than one P-state may affect a single P-state by promoting one artifact. A class of P-states is represented as a P-state without any connections on its condition boxes.

An arrow from a P-state class to an instance of a P-state represents the instantiation. The arrowhead is on the P-state instance side. An arrowhead at the beginning of the arrow which starts from a P-state and goes to another P-state represents the fact that one P-state may evoke another P-state. Both of these arrows represent the relationships between P-states. The process modeler cannot draw them to connect the condition boxes between P-states. They only connect to the edges of the P-state that do not belong to the edges of condition boxes.

The tool can shade artifact/A-state cell if the artifact reaches the defined A-state and the connecting lines disappear. The tool prints the name of activities of a P-state in the P-state boxes. The project manager on line engineer can select a zoom-out option to present the information of a P-state. In that case, they enlarge a P-state and it becomes a diagram frame analogous to the main diagram for P-state relationships. Inside this framed diagram, each activity is analogous to a P-state, the precondition box is analogous to the entrance condition, and the post-condition box is analogous to the exit condition box.

3.3.2 Artifact and Relation

Compared to the diagram for possible relationships among P-states, the diagram for the relationships among artifacts is relatively simple (see Figure 5). There is only one kind of box that is divided into two parts. A smaller part inside the box represents the name of the artifact. If it is a class of artifact, the name ends with a dot (.). The other part of the box has a list of A-states. Analogous to the arrows between a class of P-states and an instance of a P-state, the process modeler can draw an arrow from an artifact class to an instance of an artifact. The name of the artifact has a dot in it to connect the class name and the name of the instance. He can use an arrow which has the arrowhead at the beginning of the arrow to connect an artifact and its subartifacts. One artifact can have more than one subartifact, as well as more than one instance, but only one super-artifact. He can draw the artifact diagram with a P-state diagram. He can draw the dashed line between a class of artifacts or an instance of an artifact and a condition cell of a P-state. This is only by the request of a process modeler, and he can turn it on or at off any time. The graphical diagram is developed from very simple primary elements. The dynamic on/off options offer the process modeler the ability to perform a zoom/grow/cut/query cycle when they are modeling processes.

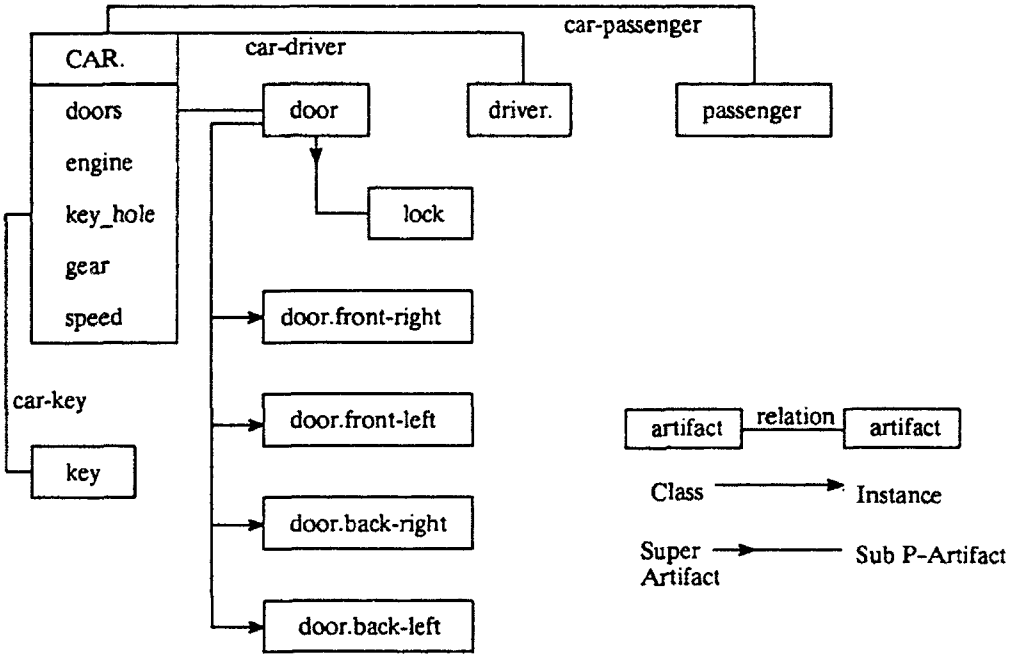


Figure 5. Artifact Relation Diagram Example

3.3.3 Possible Sequence Table for A Class of Processes

To aid in modeling a class of processes, the process modeler can use a table to show all of the possible connections between P-state condition criteria. For example, Table 12 shows the connections between entrance and exit conditions for P-state P3 and entrance and exit conditions for P-states P1 through P9. In this table state P3 uses Artifact A1 as an entrance condition, where it is connected to an exit condition on P-state P1. Similarly, P3 uses Artifact A2 as an exit condition, where it is connected to an entrance condition on P-state P4.

Table 12. Possible Sequence Table for P-State P3

Entrance Conditions									
	P1	P2	P3	P4	P5	P6	P7	P8	P9
Artifact A1	X								
Artifact A2		X							
Artifact A3									
Artifact A4									
Exit Conditions									
Artifact A1					X		X		
Artifact A2				X		X	X		
Artifact A3							X		X
Artifact A4							X	X	

Such a table can show all of the possible connections between P-states, allowing a process modeler to use the table to think about the possible sequences of the P-state and how to use the A-state to control the software process. Similar kinds of tables can be defined for other relationships, such as activities versus P-state, role versus P-state, and role versus artifacts.

3.4 A-STATE TRANSITION DIAGRAM

An A-state transition diagram is similar to a state transition diagram, where each arc represents the operation which makes the A-state change, and the node represents the A-state. Examples are in Appendix B.

4. PROCESS MODELING METHODS

4.1 BEFORE MODELING

A process modeler must make several decisions before creating a process:

- ***Granularity.*** He can describe a software process at a level detail, depending on the use of the resulting model. The easiest way to determine the granularity is by subdividing the artifacts hierarchy.

- ***Sensitivity.*** The sensitivity of a process depends on the number of A-states for a single artifact. A more sensitive process model offers the possibility for tighter control of the software process. The process modeler should keep the law of diminishing returns in mind. Beyond a certain sensitivity, the resources required to make a process more sensitive cost more than the benefits derived from such increased sensitivity.

- ***Automation.*** For process modeling, a process with automated support is different than one with manual support. An automated process needs more artifacts to record the progress of the process to support the automation, and it needs a higher degree of completeness, consistency, and formality.

- ***Formality.*** For a process to be useful, it must be well-defined and formalized. When deciding how formal to make the process, the process modeler should ask the following questions:

  - What kinds of formality support are needed for the needs of analysis?

  - What kinds of formality support are needed for process control and process management?

  - How much extra effort should be invested for a higher degree of formality?

- ***Practicality.*** The process modeler needs to decide whether a process is well-defined enough to be modeled. If a process is poorly defined and ad hoc, then it may not be practical to model the process. On the other hand, if he can model those parts of the process that are well understood, he may then be able to better define current ad hoc procedures.

- ***Scale.*** The process modeler needs to determine the size of the process and project he is modeling.

There are four aspects of software process notation that are of the greatest interest to process modelers now:

- ***Technical Development.*** Technologists and project managers use process notation to reason and plan for the integration of methods to form a continuous development process.

- ***Project Management.*** A project manager uses the process notation to highlight the landmarks of project management to ease the project manager's job, speed the process, and increase the quality.

- ***Data and Configuration Management.*** A technologist uses the process notation to ensure that the software process really works with the configuration management. There is a lot of interest in combining process notation to represent the measurement metrics for automating the process metrics data collection.

- ***Quality Assurance and Control.*** *Japan's Software Factories* contains several stories which show that, by using a formal process, technologists' and project managers' product quality can increase quality by controlling the process by which things are made.

Probably the most common reason for using process notation is to define and discuss the software process in a more precise manner. Some of the benefits are:

- ***Predictability.*** The software process is more predictable.

- ***Repeatability.*** The process can represent a repeatable process fragment.

- ***Basis for Improvement.*** The process can create a good base for process improvement.

4.2 THE MODELING PROCESS IN THE LARGE

A process modeler can choose the notation that matches the way he thinks. The following sections discuss some of the processes.

4.2.1 Bottom-Up Approach

Previous sections discussed the formal model elements and showed examples of A-states and P-states. This section describes an approach to constructing a model for a particular process. To complete such a model means that the process modeler must have some idea of the artifacts and activities that are used in the process. While he may not have a sufficiently clear view of the process at the outset, one intent of the modeling process is to help him obtain such a view. The following steps form a bottom-up approach to constructing a model that may help clarify the model under construction. As with most processes associated with software development, some backtracking is necessary in following this process.

- Define a set of artifacts according to the methodology to be used, the product to be produced, or the metrics to be gathered.

- Identify dependency and composition relations among the artifacts.

- Use the A-state model to identify a set of states for each artifact.

- Identify a set of operations for each artifact.

- Identify preconditions and post-actions for each operation.

- Identify a set of analyses that project managers and line engineers may need for making decisions in conjunction with each operation.

- Identify guidelines and methods for each operation and, where possible, translate them to analyses.
- Group the operations according to those that need to be done about the same time.
- Collect the analyses, guidelines, and operations together according to the grouping in the previous step to form a P-state.
- Identify preconditions for each P-state.

The result of this process is a set of A-states and P-states that have a flat structure. For particularly complicated models, it may be useful to introduce a hierarchical structure into the set of P-states. For example, if a process modeler is describing a complete software development process, then he might represent the design as a single P-state at the top level of the model. He may then decompose the design P-state into a hierarchy of P-states. For such cases, he adds an additional step to the modeling process. This step requires the identification of super-P-states that are groups of P-states to build a hierarchical machine. This approach is good for modeling an existing, ongoing process. The set of artifacts come from the existing project database or talking with the project engineers, or derive from the product artifacts.

4.2.2 Top-Down Approach

A top-down approach is good for modeling a process from an existing software process policy, standard, or *contract* and for modeling a process that is well understood by the process modelers.Top-down is defined here as starting at the highest level of granularity and working down to levels of increased detail and formality.

The technologist starts with the top level of the process and identifies the P-state names without pre- and post-conditions. He draws a P-state diagram and lists tentative pre- and post-conditions with the P-state. The process modeler collects the artifacts used in the pre- and post-condition to form the artifact tables. The process modeler collects a set of A-states from the pre- and post-conditions where the artifact is used for reference. He then draws an A-state transition diagram for each artifact to ensure that there is one operation for each arc in each A-state transition diagram. The operation and analysis table uses the list of operations. The process modeler treats some of the operations as P-states for a lower level of P-state. He defines the role for the lower level of operation.

4.2.3 Inside-Out Approach

An inside-out approach starts with a P-state layer, from which the process modeler then models up or down. The process modeler uses this inside-out approach mostly for modeling an essential and important part of a software process first to get the benefit of process modeling. Then, it depends on the situation of the project or company to extend the coverage of the formal process model closer to the real software process being exercised.

4.3 INSTANTIATION OF THE PROCESS

This process notation uses a dot (.) to signal both sub-of and instance-of relationships. The sub-of comes before the instance-of relationship. For example, car. represents different brands of cars; car.volvo. represents all of the Volvo cars; and car.volvo.my_car represents my Volvo car. This is true

for all of the artifacts, P-states, analyses, operations, and roles. The resolution in the implementation of this process notation is:

- If the string ends with a dot, it is the name of a class; otherwise, it is the name of an instance.
- From left to right get the character string before next dot, and call it S1.
- Get the second character string before the next dot and call it S2.
- Try to look for the definition of S2 and check if S1 is sub-of S2.
- If yes, then the dot between S1 and S2 is sub-of; otherwise, it is instance-of.

A set of rules govern the behavior of instantiation to maintain the robustness and interpretability of the process notation.

4.3.1 Instantiation Slots for a P-state

The project manager or line engineer can use eight legal specifying mechanisms to instantiate a P-state:

1. ***Attach Instance Name.*** To specify an instance of a particular class name, the process modeler writes the class name followed by a dot, followed by the instance name.
2. ***Add Operations.*** The process modeler may add more operations to the P-state.
3. ***Role Assignment.*** The process modeler may change the assignment of process agent to role.
4. ***Add Analysis.*** The process modeler may add more analysis to the P-state.
5. ***Add Operation.*** The process modeler may add more operations to the P-state.
6. ***Operation Instantiation.*** An instance of the predefined operation class represents the possible change of the P-state.
7. ***Artifact Instantiation.*** An instance of the predefined artifact class represents the possible changes of the P-state.
8. ***Add Pre-/Post-Condition.*** The process modeler may add more pre- and post- conditions to govern the relation of all the other mechanisms.

4.3.2 Instantiation Slots for an Operation

There are three legal specifying mechanisms that project managers and line engineers can use to instantiate an operation:

1. ***Role Assignment.*** The process modeler may change the assignment of process agent to role.
2. ***Artifact Instantiation.*** An instance of an artifact class, predefined as an object of the operation, represents the possible changes of the operation.
3. ***Add Pre-/Post-Condition.*** The process modeler may add more pre- and post-conditions to govern the relation of all of the other mechanisms.

4.3.3 Instantiation for an Artifact

The project manager or line engineer can only instantiate an artifact by attaching an instance name to the class name followed by a dot.

4.3.4 Instantiation for a Role

The project manager or line engineer can only instantiate a role by adding more process agents or roles in the list of process agents or subroles.

4.4 PROCESS MODELING UNDER DIFFERENT SITUATIONS

4.4.1 Modeling a Class of Processes

A class of processes is in terms of a P-state class. The P-state may have one or more operations. The process modeler represents all of the different possible processes in this class as all of the possible instantiations. Modeling an existing process can start with a set of predefined questionnaires. The answers represent the information necessary for the process modeling. The existing SEI Software Process Assessment questions are a good candidate for being extended to serve this purpose.

It is difficult to summarize a process class from information. The process modeler must always use data from actual use to calibrate and fine-tune the model. It is not necessary that all operations, artifacts, analyses and roles used to form a process class be classes themselves. Each operation, analysis, artifact, and role may be an instance of a class or its own class.

Creating a new model of processes is always difficult. The process modeler should start with a top-down approach. Even then, it is always good to model a new process before starting to use it. The process modeler should loosen his pre- and post-condition when he starts to model a new process. He should try to reach the most detailed level as soon as possible if his intention is automation. He should model some of the more unclear steps as a manual operation and wait until he obtains more information, then model it in more detail.

Modeling an old process model that already has a formal process model is always easier. Creating a parallel path from one A-state to the other A-state allows the ongoing process to co-exist with the new process which has already adopted the new process. For example, an artifact has three A-states, X, Y and Z. Operations op1 and op2 promote the artifact from X to Y and then from Y to Z. A new process can have a new operation, op3, which promotes the artifact from X to Z directly in one operation.

During the transition period, some parts of the project may use the old process and some may move to the new one. The partial order of the A-state makes this easy.

Modeling an instance of a process is easier from the top down. This approach starts with the highest level of the P-state and instantiates the operation, artifact, and next level of P-state, then goes down to the lowest level. If the process modeler cannot model some of the subprocesses, operations, or P-state from the instantiation, then he has three alternatives. He may copy the P-state or operation definition and edit it to the definition which is appropriate for the project or he can create a new definition for the project. He can also change the definition of the process class to accommodate the new need of the project.

4.5 OPPORTUNISTIC AND PROCEDURAL MODELING

Since there are close ties among artifacts, A-states, and P-states through the pre- and post-condition, it is possible to support opportunistic process modeling with the tracing and reasoning among artifacts, A-states, and P-states. The process modeler can start from any information he has for the process. He then puts the information in the definition table. The opportunistic process modeling process is a process to fill in as much as possible for all of the cells in the definition tables and then cross-checking among artifacts, A-states, P-states, and pre- and post-conditions:

- If there is no operation to promote an A-state to the next A-state, then the process modeler must create one for it.
- For all of the artifact listed in the pre- and post-conditions, he must check if there is a artifact definition table for each of them. If not, he must create one for it.
- For all of the relations used in the definition of the A-state and pre- and post-conditions, he must determine whether a relation table exists for it.

Using this method, he can iterate until he reaches all of the reasoning paths and fulfills the completeness and consistency. This is very a typical opportunistic process. The process modeler uses different clues to model the process.

Procedural modeling is more like the processes described for top-down, bottom-up, and inside-out. There is a sequence of steps to follow. It is not necessary to follow just one of them. The process modeler may combine them to model a process.

4.6 MODELING SUPPORT

Since there are formal mathematics behind the process notation, algorithms can be developed for helping process modelers. The database support for the algorithms execute the completeness- and consistency-checking and tell the process modeler what to do. At the same time, the process modeling support implemented for this process notation can also copy the overlap information from one place to another. This process notation is very discrete and separable. This increases the difficulty of modeling without the help of tools. The underlying theory not only helps manual process modeling, but can also be used to implement tools to help process modeling as well as debugging tools.

There is some experience in process modeling using this process notation. In terms of how and when to use P-states, how to derive checklist from a P-state, and how to use a formal process notation in an organization.

4.6.1 Reasons for Creating a P-State

A process modeler can create a P-state using all of the possible semantics of the software processes. In general, it is just a way of abstracting a set of operations into one operation. The following are examples:

1. ***By Artifact and A-state.*** The result of this kind of abstracting shows that an artifact listed in the entrance and exit conditions of a P-state and the A-state in the exit condition is a more complete A-state than the A-state of the same artifact listed in the entrance condition. This is common for the P-state which the process modeler creates for managing the progress of a project only and does not reflect the technical aspect of the process.

2. ***By Role or Organization.*** The result of this kind of modeling shows that only the same role can evoke most of the operation in the same P-state. These kinds of P-states are found when modeling a process involving clients, contractors, and subcontractors. This type of P-state is also found in the very low level of the technical process model.

3. ***By Operation.*** To increase the portability of the process model, it is necessary to have one version of a P-state that contains only the manual operations and another version of P-state that only contains the automatic operations. The process modeler creates some of the P-states in this way because the set of operations take special kinds of resources, e.g., machines, rooms, security.

4. ***By management.*** The process modeler uses a P-state to represent a special process state which is of interest to the manager. They may be some landmarks along the process, some critical point of the project, or deadlines.

5. ***By Measurement.*** The process modeler can also use a P-state to separate the session of measurement. In this kind of P-state, there is only analysis and no operations.

6. ***By Terms of Contract.*** This is especially common in multiple subcontractor cases. The process modeler uses a P-state to represent the finishing and starting point of two different subcontractors. The artifacts listed in the entrance and exit condition are the outputs and inputs of the two contractors.

7. ***By SEI Process Maturity Level.*** When the process modeler uses the formal process notation to model the process improvement process based on SEI's process maturity levels, each maturity level is one P-state.

Additional instances in which a process modeler may use a P-state include:

- To represent repeatable process fragments. The P-state is the construct in this process notation for representing repeatable process fragments.

- Representing a hierarchical process architecture for a large or complex process. For modeling a large process, the process modeler can use a P-state to group operations together and treat them as one operation. He can also group this operation with other operations to form another P-state and treat this group as one operation as well. In this way, he can represent a large, complex process.

4.6.1.1 Mapping the Formal Process Model to a Checklist

In practice, a process modeler usually uses checklists as a record for the progress of activities. As soon as he models a process using this process notation, he can derive a set of checklists from the formal process model. The mapping from the P-state to the checklist is as follows:

- Every P-state has a checklist.

- Every operation is an entry of a checklist.

- Every manual operation has a working data sheet.

- Every automatic operation has a program/tool.

4.6.1.2 Using the Notation in an Organization

To adopt the formal process notation in an organization:

1. Start with a step-by-step approach, using whatever is useful, possible, and beneficial. Increase the speed of adoption when the company begins to gain something from the notation and has room to adopt more.

2. Increase the granularity with the understanding of software process. It is not necessary to describe all the details of the process in the initial model.

3. Combine the formal process notation with the use of an existing process notation. Between English and mathematics, there are different degrees of formality and notations. Start with one notation first and then combine other notations for process representation if the project benefits from the notations.

4. Start with the basic routine process because they are repeated more times than others.

5. IMPLEMENTATION OF A PROCESS NOTATION

5.1 DESIGN AND IMPLEMENTATION OF PROCESS-CENTERED ENVIRONMENTS

5.1.1 Implementation for Process Notation

To implement a process-centered environment for process notation, the following must be done:

- Maintain artifact states.
- Guide the sequences of decisions permitted by the P-state transitions.
  - Prevent forbidden transitions; e.g., a module cannot be done before its interface specification is done.
  - Suggest possible transitions; e.g., identify modules ready to be coded.
- Provide specified operations and analyses and provide help in performing operations that are left as manual.
- Integrate a toolset around a process.

5.2 USING THE MODEL TO BUILD SOFTWARE PROCESS MANAGEMENT TOOLS

5.2.1 Building Process-Centered Software Environment Tools

A two-level model of a software process can be viewed both as a specification for the process and as a specification for a process-centered software environment to be used to support the process. The model specifies the data upon which the tools must operate (i.e., the set of software artifacts and their interdependencies), the operations to be performed by the tools, and the effects of those operations (i.e., the state changes caused by invoking the tools). Accordingly, the technologist may use a particular model to explain the process to software developers who use it, to tool developers who build tools to support it, or to tool integrators who integrate a set of existing tools to support the process.

The type of specification that the technologist uses in describing operations and analyses is left to the discretion of the modelers and environment implementors. There are four types of specification which the technologist can choose through the implementation of a process notation. They are:

- Tools in terms of a programming library in a high-level programming language.
- Standalone COTS tool.

- Operating system command-level scripts, e.g., shell script.
- Tool integration scripts, e.g., Software BackPlane's script.

The implementation reads the process notation and generates the software environment in terms of tool integration and activities coordination. For different projects, methods used, purposes and users, the process modeler uses different types at the same time. To describe the process to software developers, he may use an informal prose specification or a formal specification that describes effects in terms of state changes on composite artifacts. As an example of the latter, the effect of using an editor to create an abstract interface might be specified as:

```
state_of(abstract_interface) := CREATED
```

To describe the services offered by the editor to the software developer, the process modeler may provide a reference to the user's manual for the editor. To describe to the interface provided by the editor to a tool builder, he may reference an interface specification for the editor. To specify the tool to be built, he may reference a requirements specification for the editor. The modeler may use any combination of such references and specifications, as appropriate.

In particular, the implementor of the tool must execute actions that are implicitly specified by the model. For example, in cases where there are dependencies among design artifacts, a state change in one artifact may cause state changes in other artifacts. Furthermore, there may be informal parts of the model that the formal model does not capture well, e.g., specifying methods for resolving contending requests to update the same design artifacts by several different designers. Such requests may arise directly (e.g., when several designers are simultaneously attempting to manipulate the same design artifact) or indirectly (e.g., when several designers simultaneously change the states of dependent design artifacts).

One way to make implicit actions more explicit for the purposes of implementation, or to specify conflict resolutions, is to provide a way of specifying actions to be taken on the occurrence of certain types of events. One useful such class is a post-action accompanying a P-state exit condition. For example, on leaving the P-state Abstract_Interface_Design, the following post-action might be taken:

```
if (state_of(abstract_interface) = DONE)
notify(reviewers_of(abstract_interface))
```

5.2.2 Interface Between Process Model and Software Environment

The Consortium designed the process notation to precisely represent the interface between the process model and the process-centered software environment implementation. Figure 6 shows the structure of the interfaces between the process model and the implementation of the process-centered software environment. This is a world. Part of this world deals with software development and is called the modeled world. There are two views of the modeled world:

- ***Information Processing View.*** This view divides the modeled world into information and the mechanisms to process data (information).
  - ***Information World.*** In this world, everything is an artifact. The artifact contains a name, a set of A-states, and a reference to the object in the software environment world.

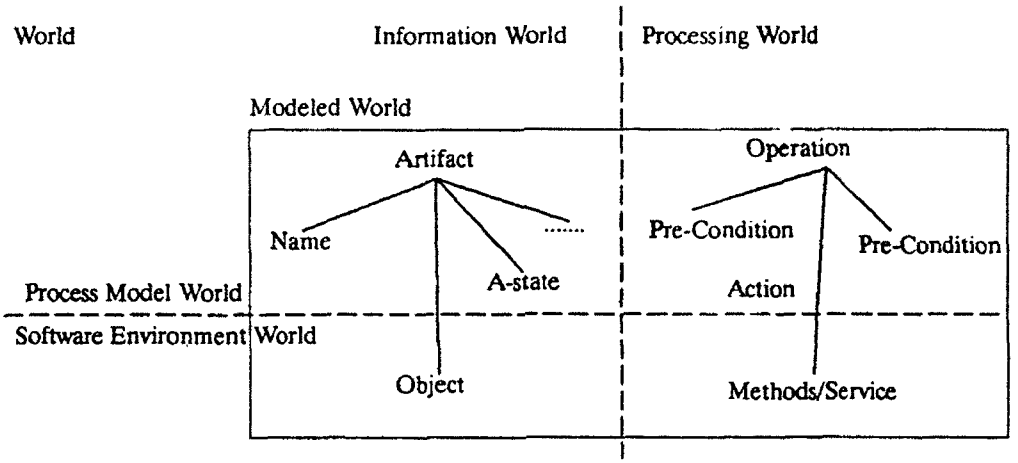


Figure 6. Structure of the Interface Between a Process Model and the Process-Centered Software Environment

- ***Processing World.*** In this world, everything is an operation. An operation contains a precondition, a post-condition, and a reference, called action, to some methods in the software environment world which actually do the job.

- ***Software Environment View.*** This view divides the modeled world into a territory in which the process modeler lives and a territory in which a software environment toolsmith lives.

  - ***Process Model World.*** In this world, people only pay attention to the salient features of the process such as the operations that they need to complete and the artifacts that they need to generate. Everything is an artifact or an operation. The sequence of doing something is always governed by pre- and post-conditions. From time to time, they need to do some analysis to know what is going on to make a decision of what to do next. One way to package the routine sequence events is to form a P-state.

  - ***Software Environment World.*** People in this world are well-equipped with object-oriented design/implementation, configuration management, tool integration knowledge, and techniques. They contract with people in the process model world through the requirement, which is written in the process notation.

Based on the interface model described above, the process notation is designed so that it:

- Lets the process modeler pay most or all of his attention to the process modeling by:

  - Defining the basic elementary artifact classes on top of the set of object classes listed in the interface/contract with the software environment world, or on top of the set of predefined object classes offered by the process notation.

  - Defining the composite artifact classes on top of the elementary artifact classes.

  - Defining operations/actions and analysis on top of the methods/service offered by the notation and the process-centered software environment implementation.

  - Defining the contract for some environment implementation if necessary.

- Parallels the process-centered software environment implementation with process modeling by taking the notation as contract.
- Integrates a multiple environment implementation for a single software project's process.

5.2.3 Suggestions for Implementing Forms and Graphic Diagrams

Following are some suggestions that the process modeler may incorporate into an environment that processes the forms or graphic diagrams discussed in Section 3.

The implementation should be able to generate all of the forms for a process model based on the information captured in the graphical diagram. The forms do not need to be filled in totally. At the same time, the implementation should be able to prompt the line engineer for the information which is lacking. The line engineer should be free to move back and forth between graphical diagrams and form editors.

It is very difficult to draw all the existing P-state classes, P-state instances, and the process model in one diagram. The implementation should offer the ability to draw the full diagram if the project manager or line engineer so choose. The implementation should also offer them the ability to:

- Connect only the condition of the artifact/A-state cell for the selected artifacts.
- Turn on/off the information for the class of P-states.
- Turn on/off the information for the super- and sub-P-state relationship.
- Show only the steps to reach a particular A-state of one single artifact.
- Show only the steps of opening one single P-state.
- Allow some of the P-states to zoom out.
- Combine some of the options.

The use of these options simplifies the resulting graphical diagram.

It is still possible to have a process so large that the artifact diagrams are too complex. The implementation should offer the following options for simplifications:

- Turn on/off information for one or more subartifacts.
- Turn on/off relations of super- and subartifacts for one or more artifacts.
- Query the P-state diagram that is related to one or more artifacts from artifact diagrams.
- Trace the subartifact relation.
- Trace the superartifact relationship.
- Trace the instance artifact relationship.
- Trace the parent artifact relationship.

5.2.4 Integrating CASE Tools

The modeler can use process model to integrate different software development tools to support a software process. He can do this in the following steps:

- For each operation and analysis in each P-state, select a set of commercially available tools that he can use to perform the operations and analyses.
- Design the interfaces that permit the tools to access the software artifacts and their states.
- Implement actions as invocations of an existing, commercially available tool using the interfaces.

Using this procedure, the model becomes an instrument for designing a common tool environment. Some tool integration tools (e.g., Atherton's Software BackPlane) are good for implementing a process notation because they can use existing executable binary of commercial software tools, UNIX shell scripts, program source code, and tool integration tool's scripts.

This process notation allows the representation mechanism to express all of the four possible interfaces to the process-centered software environment implementation.

5.2.5 Questions That a Tool Can Answer

A tool developed to support the model can also answer questions such as:

- What is the difference between version X and version Y of the software?
- What modifications have been made since a design artifact reached a particular state, or since a particular P-state was last visited?
- What modules are affected by a particular modification or batches of changes or modifications? Who modified them, and for what reason?
- What remains to be done to complete a particular state?
- What are the states of various design artifacts?

This page intentionally left blank.

APPENDIX A. PROCESS DEFINITION FORM TEMPLATE SET

A.1 ARTIFACT DEFINITION FORM TEMPLATE

Artifact Definition Form			
Name			
Synopsis	(English description here)		
Complexity Type	(Elementary or Composite)		
Data Type	(Name of object class, predefined or user-defined)		
A-State List			
A-State-Name	Put condition lists over A-state of dependent artifacts here, if artifact is composite artifact.	(English explanation here)	
Subartifact List			
First Subartifact Name	(Name)	(English explanation here)	
Relations List			
Name of relation	(English explanation here)		
	Type of relation (predefined or user-defined)	Direction (to or from)	Name of related artifacts

A.2 RELATION DEFINITION FORM TEMPLATE

Relation Definition Form			
Name			
Synopsis	(English description here)		
Possible Artifact Pair List			
From		To	
From		To	
From		To	

A.3 PROCESS-STATE DEFINITION FORM TEMPLATE

Process-State Definition Form	
Name	
Synopsis	(English description here)
Main Role	

Process-State Definition Form		
Entrance Condition		
Artifacts List		
Information Artifacts		
Operation List		
	Name	
	Description	
Analysis List		
	Name	
	Description	
Post-Action List		
	Description	
	Condition	
	Action Function	
Exit Condition		

A.4 OPERATION DEFINITION FORM TEMPLATE

Operation Definition Form		
Name		
Synopsis	(English description here)	
Precondition List		
	Operation Type	(Manual, tool, function library or state_control)
	Artifact	
	Action	
	Role List	
Post-Condition		

A.5 ANALYSIS DEFINITION FORM TEMPLATE

Analysis Definition Form		
Name		
Synopsis	(English description here)	
Information Artifacts		
Analysis Function		
	Analysis Type	(Management, quality, consistency, process status, or user_defined)
	Artifact	
	Analysis Function	
	Role List	
	Trigger Type	(State_entrance, state_exit, user, periodical, or operation)
	Result Type	(File, data_structure, diagram, table, text_report, or form)
	Action Type	(Prompt message, save to file, or mail to role)

A.6 ACTION DEFINITION FORM TEMPLATE

Action Definition Form	
Name	
Synopsis	(English description here)
Type	(function_library, shell_script, tool)
Source Path Name	
Input Object Class	
Input Path Name	
Output Object Class	
Output Path Name	

A.7 ROLE DEFINITION FORM TEMPLATE

Role Definition Form	
Name	
Synopsis	(English description here)
Member List	
Activity List	

This page intentionally left blank.

APPENDIX B. A FORMAL MODEL OF A CAR-DRIVING PROCESS CLASS

B.1 ARTIFACT DEFINITION

B.1.1 CAR.

B.1.1.1 Artifact Definition Form: Car.

Artifact Definition Form		
Name	Car.	
Synopsis	This is the artifact which represent a class of cars.	
Complexity Type	Composite	
Data Type	(car_c, user-defined)	
A-State List		
parked	((state_of(car.engine) = off) (state_of(car.gear) = park) (state_of(car.speed) = stand))	Car is not moving, and engine is not running.
initiated	((state_of(car.engine) = on) (state_of(car.key_hole) = has-key) (state_of(car-driver(car.)) = in-car) (state_of(car.gear) = drive) (state_of(car.speed) = stand))	Car is not moving, but the engine is running.

Artifact Definition Form		
moving	((state_of(car.engine) = on) (state_of(car.key_hole) = has-key) (state_of(car-driver(car.)) = driving) ((state_of(car.gear) = drive) or (state_of(car.gear) = reverse)) ((state_of(car.speed) = stand) or (state_of(car.speed) = slow) or (state_of(car.speed) = medium) or (state_of(car.speed) = high)))	Car is moving forward or backward.
Subartifact List		
	doors	The four doors of a car.
	engine	The engine of a car.
	key_hole	The key hole of a car.
	gear	The gear of a car.
	speed	The speed of a car.
Relations List		
car-key	This is the relation between a car and a key.	
car-driver	This is the relation between a car and a driver.	

B.1.1.2 Artifact State Transition Diagram: Car.

Figure 7 shows the A-state transition diagram for the car.

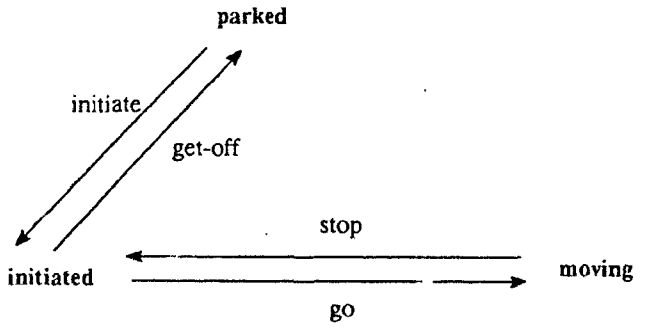


Figure 7. A-State Transition Diagram for Car.

B.1.2 Doors

B.1.2.1 Artifact Definition Form: Doors

Artifact Definition Form		
Name	doors	
Synopsis	The artifact is a composite artifact of all four doors of a car.	
Complexity Type	Composite	
Data Type	(door_c, user-defined)	
A-State List		
open	(for_all x = sub_ artifact(doors.) there_ exist_one (state_of(x) = open))	The door is opened.
closed	(for_all x = sub_ artifact(doors.) (state_of(x) = closed))	The door is closed.
Subartifact List		
	door.front-right	The front right door of a car.
	door.front-left	The front left door of a car.
	door.back-right	The back right door of a car.
	door.back-left	The back left door of a car.
Relations List	None	

B.1.2.2 Artifact State Transition Diagram: Doors

Figure 8 shows the A-state transition diagram for doors.

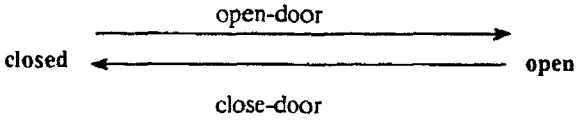


Figure 8. A-State Transition Diagram for Doors

B.1.3 Engine

B.1.3.1 Artifact Definition Form: Engine

Artifact Definition Form	
Name	engine
Synopsis	The engine of a car.
Complexity Type	Elementary
Data Type	(c_engine user-defined)
A-State List	

Artifact Definition Form		
on		The engine is running.
off		The engine is not running.
Subartifact List	None	
Relations List	None	

B.1.3.2 Artifact State Transition Diagram: Engine

Figure 9 shows the A-state transition diagram for engine.

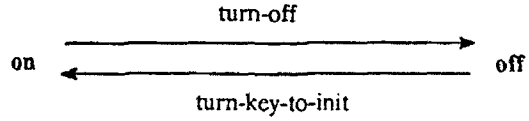


Figure 9. A-State-Transition Diagram for Engine

B.1.4 KEY_HOLE

B.1.4.1 Artifact Definition Form: Key_hole

Artifact Definition Form		
Name	key_hole	
Synopsis	The keyhole to engage the electrical system of the car to start it running.	
Complexity Type	Elementary	
Data Type	(c_key-hole, user-defined)	
A-State List		
empty		There is no key in the keyhole.
has-key		There is a key in the keyhole.
Subartifact List	None	
Relations List	None	

B.1.4.2 Artifact State Transition Diagram: Key_hole

Figure 10 shows the A-state transition diagram for key_hole.

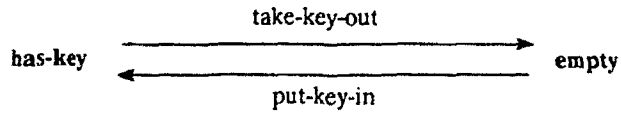


Figure 10. A-State Transition Diagram for Key_hole

B.1.5 Gear

B.1.5.1 Artifact Definition Form: Gear

Artifact Definition Form		
Name	gear	
Synopsis	It represents the gear box of a car.	
Complexity Type	Elementary	
Data Type	(c_gear, user-defined)	
A-State List		
park		The gear is in park.
neutral		The gear is in neutral.
reverse		The gear is in reverse.
drive		The gear is in drive.
Subartifact List	None	
Relations List	None	

B.1.5.2 Artifact State Transition Diagram: Gear

Figure 11 shows the A-state transition diagram for gear.

B.1.6 Speed

B.1.6.1 Artifact Definition Form: Speed

Artifact Definition Form		
Name	speed	
Synopsis	It represents the speed of the car in the speed range.	
Complexity Type	Elementary	
Data Type	(c_speed, user-defined)	
A-State List		
stand		The car is not moving.
slow		The car is moving less than fifteen miles per hour.
medium		The car is moving more than fifteen miles per hour but less than forty miles per hour.
high		The car is moving more than forty miles per hour.
Subartifact List	None	
Relations List	None	

B.1.6.2 Artifact State Transition Diagram: Speed

Figure 12 shows the A-state transition diagram for speed.

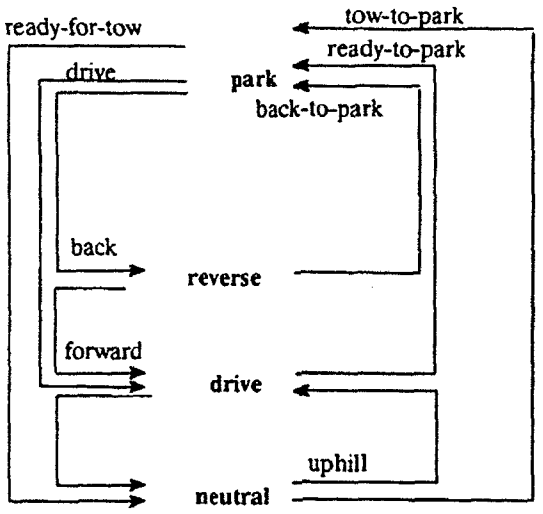


Figure 11. A-State Transition Diagram fo: Gear

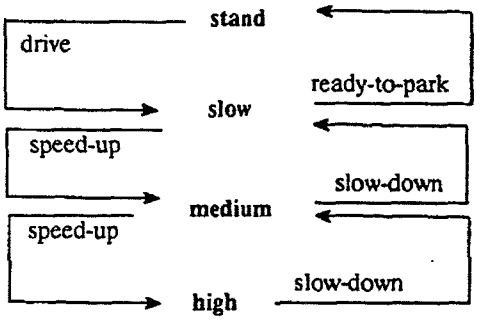


Figure 12. A-State Transition Diagram for Speed

B.1.7 Door.

B.1.7.1 Artifact Definition Form: Door.

Artifact Definition Form		
Name	door.	
Synopsis	The four or two doors of a car.	
Complexity Type	Composite	
Data Type	(c_door, user-defined)	
A-State List		
open		The door is opened.
closed		The door is closed.
locked	(state_of(door.lock) = locked)	The door is locked.
unlocked	(state_of(door.lock) = unlocked)	The door is unlocked.
Subartifact List		

Artifact Definition Form		
	lock	The lock of a door.
Relations List	None	

B.1.7.2 Artifact State Transition Diagram: Door.

Figure 13 shows the A-state transition diagram for door.

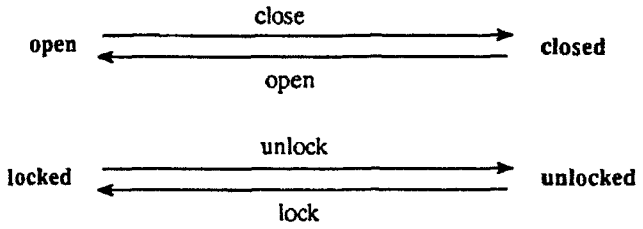


Figure 13. A-State Transition Diagram for Door.

B.1.8 Lock

B.1.8.1 Artifact Definition Form: Lock

Artifact Definition Form		
Name	lock	
Synopsis	A kind of lock.	
Complexity Type	Elementary	
Data Type	(c_lock, user-defined)	
A-State List		
locked		The door is locked.
unlocked		The door is unlocked.
Subartifact List	None	
Relations List	None	

B.1.8.2 Artifact State Transition Diagram: Lock

Figure 14 shows the A-state transition diagram for lock.

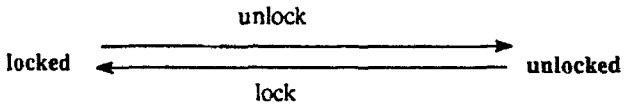


Figure 14. A-State Transition Diagram for Lock

B.1.9 Driver.

B.1.9.1 Artifact Definition Form: Driver.

Artifact Definition Form	
Name	driver.
Synopsis	The artifact captures the state of a driver.

Artifact Definition Form		
Complexity Type	Elementary	
Data Type	(c_driver, user-defined)	
A-state List		
in-car		The driver is in the car.
out-car		The driver is out of the car.
driving		The driver is in the car and driving.
Subartifact List	None	
Relations List		
driver-key	It represents the binding between driver and key.	
car-driver	It represents the binding between driver and car.	

B.1.9.2 Artifact State Transition Diagram: Driver.

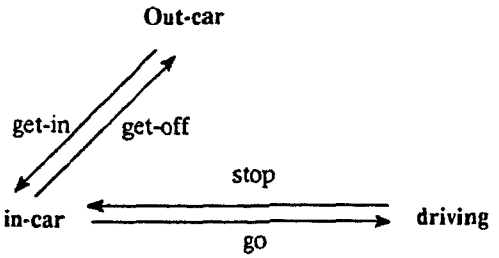


Figure 15. A-State Transition Diagram for Driver.

Figure 5 shows the A-state transition diagram for driver.

B.1.10 Passenger.

B.1.10.1 Artifact Definition Form: Passenger.

Artifact Definition Form		
Name	passenger.	
Synopsis	It represents a passenger of a car.	
Complexity Type	Elementary	
Data Type	(c_passenger, user-defined)	
A-state List		
out-car		The driver is not in the car.
in-car		The driver is in the car.
buckled		The driver has the seat belt on.
unbuckled		The driver does not have the seat belt on.
Subartifact List	None	
Relations List		
passenger-car		

B.1.10.2 Artifact State Transition Diagram: Passenger.

Figure 16 shows the A-state transition diagram for passenger.

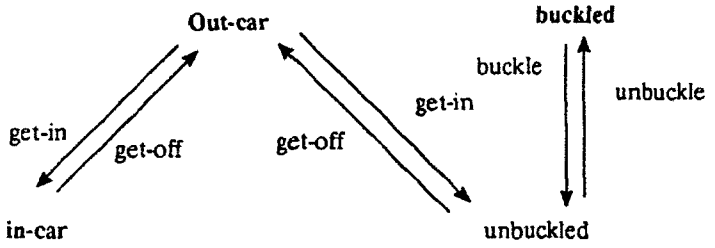


Figure 16. A-State Transition Diagram for Passenger.

B.1.11 Key.

B.1.11.1 Artifact Definition Form: Key.

Artifact Definition Form		
Name	key.	
Synopsis	It represents a key of a car.	
Complexity Type	Elementary	
Data Type	(c_key, user-defined)	
A-state List		
on_table		The key is put on a table.
in_pocket		The key is put in a pocket.
in_purse		The key is put in a purse.
in_key_hole		The key is inserted in the keyhole of a car.
electricity		The key is used to turn the electricity of a car.
drive		The key is being used to drive.
init		The key is turned to the ignition position.
Subartifact List	None	
Relations List	None	

B.1.11.2 Artifact State Transition Diagram: Key.

Figure 17 shows the A-state transition diagram for key.

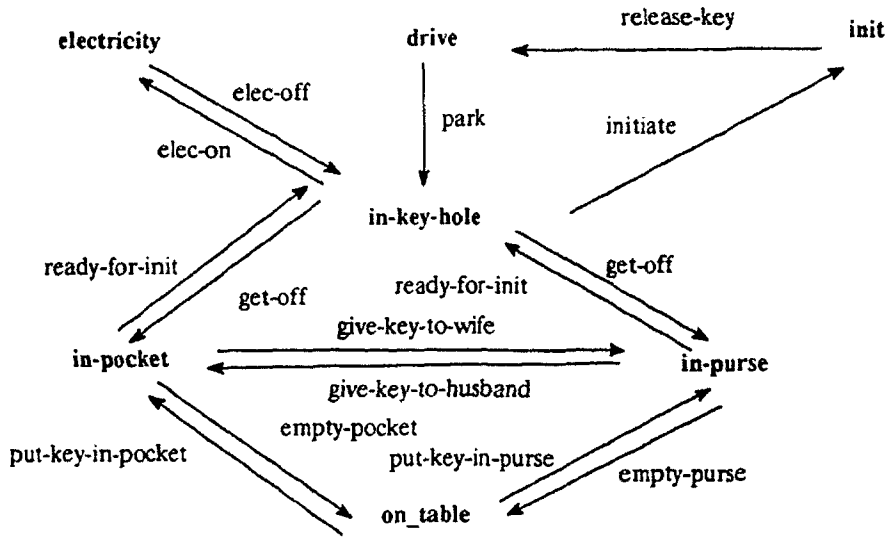


Figure 17. A-State Transition Diagram for Key.

B.2 RELATION DEFINITION FORMS

B.2.1 CAR-KEY

Relation Definition Form			
Name	car-key		
Synopsis	This relationship specifies that a key is related to a car by the fact that, when it is put into the ignition switch and turned, it causes the cars engine to start running.		
Possible Artifact Pair List			
From	car	**To**	key
From	key	**To**	car

B.2.2 CAR-DRIVER

Relation Definition Form			
Name	car-driver		
Synopsis	This relation binds a car to a driver when he drives.		
Possible Artifact Pair List			
From	car	**To**	driver
From	driver	**To**	car

B.2.3 DRIVER-KEY

Relation Definition Form			
Name	driver-key		
Synopsis	This relates the holder of the key to the key.		
Possible Artifact Pair List			
From	driver	To	key
From	key	To	driver

B.2.4 PASSENGER-CAR

Relation Definition Form			
Name	passenger-car		
Synopsis	This relates the passenger to the car he is in.		
Possible Artifact Pair List			
From	passenger	To	car
From	car	To	passenger

B.3 OPERATION DEFINITION FORMS:

B.3.1 INITIATE

Operation Definition Form		
Name	initiate	
Synopsis	The driver gets into the car, gets the key out of his pocket, inserts it into the key-hole, turns it to initiate the car, and then releases the key. The key will automatically return to the drive position.	
Precondition List	(state_of(car.) = parked)	
	Operation Type	Manual
	Artifact List	Car.
	Action	None
	Role List	driver.
Post-Condition	(state_of(car.) = initiated)	

B.3.2 GET-OFF

Operation Definition Form		
Name	get-off	
Synopsis	When the engine of the car is stopped but the engine is still running, the driver turns the key to the off position, unbuckles the seat belt, and then opens the door on the driver side and gets out of the car.	
Precondition List	(state_of(car.) = initiated)	
	Operation Type	Manual

Operation Definition Form		
	Artifact List	Car.
	Action	None
	Role List	driver.
Post-Condition	(state_of(car.) = parked)	

B.3.3 Stop

Operation Definition Form		
Name	stop	
Synopsis		
Precondition List	(state_of(car.) = initiated)	
	Operation Type	Manual
	Artifact List	Car. driver.
	Action	None
	Role List	driver.
Post-Condition	(state_of(car.speed) = stand	

B.3.4 Go

Operation Definition Form		
Name	go	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Car. driver.
	Action	None
	Role List	driver.
Post-Condition		

B.3.5 Open-door

Operation Definition Form		
Name	open_door	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Doors
	Action	None
	Role List	
Post-Condition		

B.3.6 CLOSE-DOOR

Operation Definition Form		
Name	close-door	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Doors
	Action	None
	Role List	
Post-Condition		

B.3.7 TURN-OFF

Operation Definition Form		
Name	turn-off	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Engine
	Action	None
	Role List	driver.
Post-Condition		

B.3.8 TURN-KEY-TO-INIT

Operation Definition Form		
Name	turn-key-to-init	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Engine
	Action	None
	Role List	driver.
Post-Condition		

B.3.9 TAKE-KEY-OUT

Operation Definition Form		
Name	turn-key-out	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Key key_hole
	Action	None
	Role List	
Post-Condition		

B.3.10 PUT-KEY-IN

Operation Definition Form		
Name	put-key-in	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Key key_hole
	Action	None
	Role List	
Post-Condition		

B.3.11 READY-FOR-TOW

Operation Definition Form		
Name	ready-for-tow	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.12 Drive

Operation Definition Form		
Name	drive	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.13 Back

Operation Definition Form		
Name	back	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.14 Forward

Operation Definition Form		
Name	forward	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.15 UPHILL

Operation Definition Form		
Name	uphill	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.16 BACK-TO-PARK

Operation Definition Form		
Name	back-to-park	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.17 READY-TO-PARK

Operation Definition Form		
Name	ready-to-park	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.18 TOW-TO-PARK

Operation Definition Form		
Name	tow-to-park	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.19 SPEED-UP

Operation Definition Form		
Name	speed-up	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Speed
	Action	None
	Role List	
Post-Condition		

B.3.20 SLOW-DOWN

Operation Definition Form		
Name	slow-down	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.21 Close

Operation Definition Form		
Name	close	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.22 Lock

Operation Definition Form		
Name	lock	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.23 Unlock

Operation Definition Form		
Name	unlock	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.24 GET-IN

Operation Definition Form		
Name	get-in	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.25 GET-OFF

Operation Definition Form		
Name	get-off	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.26 BUCKLE

Operation Definition Form		
Name	buckle	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.27 UNBUCKLE

Operation Definition Form		
Name	unbuckle	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.28 ELEC-OFF

Operation Definition Form		
Name	elec-off	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.29 ELEC-ON

Operation Definition Form		
Name	elec-on	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.30 READY-FOR-INIT

Operation Definition Form		
Name	ready-for-init	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.31 PUT-KEY-IN-POCKET

Operation Definition Form		
Name	put-key-in-pocket	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.32 EMPTY_POCKET

Operation Definition Form		
Name	empty-pocket	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.33 PUT_KEY_IN_PURSE

Operation Definition Form		
Name	put-key-in-purse	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.34 EMPTY_PURSE

Operation Definition Form		
Name	empty_purse	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.35 GIVE_KEY_TO_WIFE

Operation Definition Form		
Name	give-key-to-wife	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.36 GIVE_KEY_TO_HUSBAND

Operation Definition Form		
Name	give-key-to-husband	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.3.37 INITIATE

Operation Definition Form		
Name	initiate	
Synopsis		
Precondition List		
	Operation Type	Manual
	Artifact List	Gear
	Action	None
	Role List	
Post-Condition		

B.4 PROCESS-STATE DEFINITION FORM

B.4.1 INITIATE.

Process-State Definition Form	
Name	initiate.
Synopsis	
Main Role	driver.
Entrance Condition	((state_of(car.engine) = off) (state_of(car-driver(car.)) = out_car) (state_of(car.gear) = park) (state_of(car.speed) = stand) (state_of(car.speed) = stand) (state_of(key.)= in_pocket) (state_of(key.) = in_purse))
Artifacts List	Car. key.
Information Artifacts	passenger. driver. back-mirror car-health-light.
Operation List	

Process-State Definition Form		
	Name	elec_on
	Description	
	Name	elec_off
	Description	
	Name	go
	Description	
	Name	open_door
	Description	
	Name	close_door
	Description	
	Name	turn_key_to_init
	Description	
	Name	release_key
	Description	
	Name	put_key_in
	Description	
	Name	take_key_out
	Description	
	Name	drive
	Description	
	Name	lock
	Description	
	Name	unlock
	Description	

Process-State Definition Form		
Analysis List		
	Name	check-passenger
	Description	
	Name	check-back-mirror
	Description	
	Name	check-seat-belt
	Description	
Post-Action List		
	Description	If engine oil is low, add engine oil.
	Condition	state_of(car-health-light.engine-oil) = on
	Action Function	add-engine-oil(car.)
	Description	If electricity is low, wait for five minutes.
	Condition	state_of(car-health-light.electricity) = low
	Action Function	wait(5 min)
	Description	If coolant is low, add coolant.
	Condition	state_of(car-health-light.coolant) = on
	Action Function	add-engine-coolant(car.)
Exit Condition	((state_of(car.engine) = on) (state_of(key_hole) = has_key) (state_of(car-driver(car.)) = in_car) (state_of(car.gear) = drive) (state_of(car.speed) = stand))	

B.4.2 Drive.

Process-State Definition Form		
Name	drive.	
Synopsis		
Main Role	driver.	
Entrance Condition	((state_of(car.engine) = on) (state_of(car_driver(car.)) = in_car) (state_of(car.gear) = drive) (state_of(car.speed) = stand))	
Artifacts List	car.	
Information Artifacts	passenger. driver. right-mirror left-mirror back-mirror speed-meter engine-heat-meter gas-meter	
Operation List		
	Name	drive
	Description	
	Name	back
	Description	

Process-State Definition Form		
	Name	forward
	Description	
	Name	uphill
	Description	
	Name	ready_for_tow
	Description	
	Name	speed_up
	Description	
	Name	slow_down
	Description	
	Name	open_door
	Description	
	Name	close_door
	Description	
	Name	lock
	Description	
	Name	unlock
	Description	
	Name	buckle
	Description	
	Name	unbuckle
	Description	
Analysis List		
	Name	check-speed
	Description	
	Name	check-back-mirror
	Description	
	Name	check-left-mirror
	Description	
	Name	check-right-mirror
	Description	
Post-Action List		
	Description	If gas is running low, add gas before parking.
	Condition	state_of(gas-meter) = low
	Action Function	add_gas(car.)

Process-State Definition Form		
	Description	If engine is too hot, let it run 20 minutes before parking.
	Condition	state_of(engine-heat-meter) = too_hot
	Action Function	wait(10 min)
Exit Condition	((state_of(car.engine) = on) (state_of(car-driver(car.)) = in-car) ((state_of(car.gear) = drive) or (state_of(car.gear) = reverse)) (state_of(car.speed) = stand))	

B.4.3 Park.

Process-State Definition Form		
Name	park.	
Synopsis		
Main Role	driver.	
Entrance Condition	((state_of(car.engine) = on) (state_of(car-driver(car.)) = in-car) (state_of(car.gear) = drive) (state_of(car.speed) = stand))	
Artifacts List	car.	
Information Artifacts	mirror. passenger. tire. weather.	
Operation List		
	Name	turn_off
	Description	
	Name	get_off
	Description	
	Name	ready_to_park
	Description	
	Name	open_door
	Description	
	Name	close_door
	Description	
Analysis List		
	Name	check_air_pressure
	Description	
	Name	check_back_mirror
	Description	
	Name	check_gas
	Description	
Post-Action List		
	Description	If passenger is sleeping, wake him up.

Process-State Definition Form		
	Condition	state_of(passenger.) = sleeping
	Action Function	wake_up(passenger.)
	Description:	If it is raining, make sure there is one umbrella in the car.
	Condition	state_of(weather.) = raining
	Action Function	check_umbrella()
Exit Condition	((state_of(car.engine) = off) (state_of(car.gear) = off) (state_of(car.speed) = stand) (state_of(car-driver(car.)) = out_car))	

B.5 PROCESS-STATE DIAGRAM

Figure 18 shows the process-state diagram for the initiate, park and drive P-states.

B.6 ARTIFACT DIAGRAM

Figure 19 shows the artifact relation diagram for the artifacts of the car-driving process class.

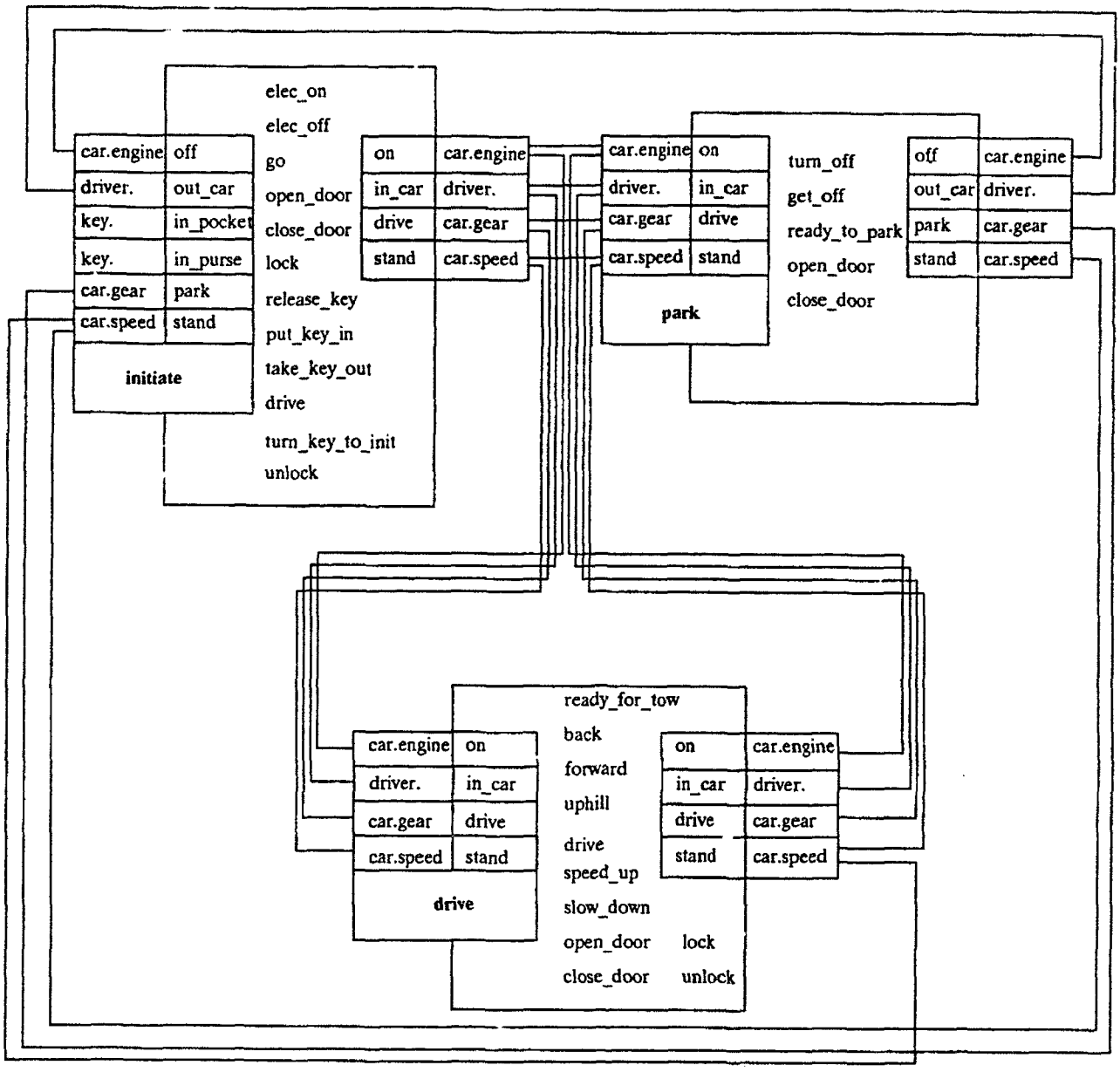


Figure 18. Graphical Representation of Car Activities and Pre- and Post-Conditions

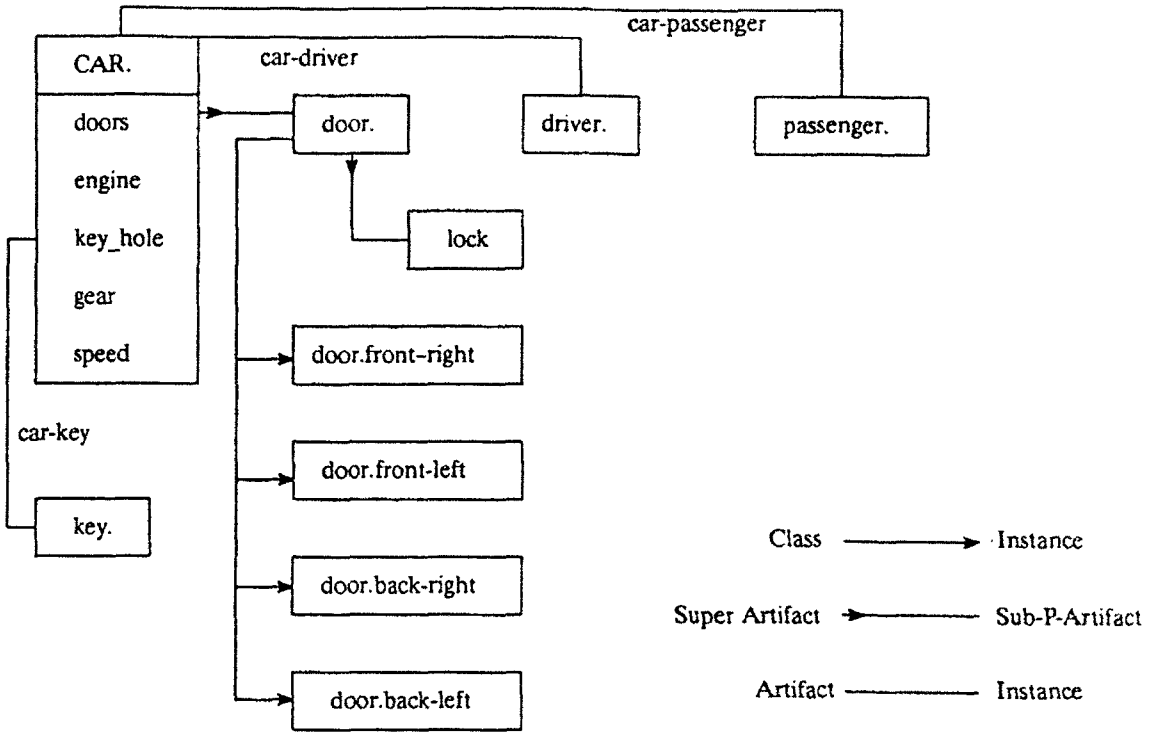


Figure 19. Artifact Relation Diagram Example: Car Driving

GLOSSARY

Accuracy	The degree to which the product produced by the process matches the intended result.
Contract	A formal agreement on a set of externally observable process enactment states and results. This agreement is a process constraint, whose violation has legal consequences (which are outside the process space unless that aspect is modeled as well).
Guidance	The use of a process definition (constraint) by an observer or process agent to provide the enacting process agent with the legal set of process step options at any point of the enactment of the observed process. This may involve process cues, process interaction, or process management.
Policy	A guiding principle; a process constraint, usually at a high level, that focuses on certain aspects of a process and influences the enactment of that process.
Precision	The degree to which the process definition completely specifies all the actions needed to produce accurate results. That is, a precisely defined process, executed with fidelity, produces an accurate result.
Predictability	An indication that either the process is intended to terminate and does terminate, or that the process is intended to be nonstop and that it does continue until terminated by a control process (or its agent).
Process	A series of actions or operations conducing to an end. A series of actions intended to reach a goal, possibly resulting in products.

Process agent	An entity (human or nonhuman) that enacts (i.e., interprets) a process definition. This entity may be a person following a process script or a hardware/software interpreter. Examples are programming language translators and interpreters, user interface language interpreters (e.g., User Interface Management System [UIMS]), CPUs that interpret a process program, or a software professional implementing a software change according to a process script.
Process architecture	A conceptual framework for incorporating process elements in consistent ways (or for signaling that the process element is incompatible with the architecture). A framework within which project-specific processes are defined.
Process constraint	A process definition describing conditions of an external event or of another process definition that are to be satisfied. Examples of process constraints are requirements, authorization, standards, and procedures. Such constraints may be included in process designs, process programs, process scripts, process architectures, or process plans.
Process control	The external influence over process enactment by other enacting processes. This influence may be driven by process evaluation and may be through control of the process enactment state, reassignment of resources, or change of process goals through process evolution.
Process definition	An instantiation of a process design for a specific project team or individual. It consists of a partially ordered set of process steps that is enactable. Each process step may be further refined into more detailed process steps. A process definition may consist of (sub)process definitions that can be concurrently enacted. Process definitions, when enactable by humans, are referred to as process scripts. Process definitions for nonhuman enactment are referred to as process programs.

Process evaluation	The analysis of observed process information and its comparison with predefined constraints. This includes both process design/definition constraints and process plan constraints. This may concern any process enactment activities.
Process evolution	The evolution of process definitions (static) as well as the evolution of enacting processes (dynamic), e.g., nonstop processes. Both static and dynamic change must be managed to ensure stability of the process and control over the process results.
Process model	A possibly partial process definition for the purpose of modeling certain characteristics of an actual process. Process models can be analyzed, validated, and, if enactable, simulates the modeled process. Process models may model at the process architecture, design, or definition level. Process models are at times used to predict process behavior. Process models themselves have an architecture, a design, and a definition.
Process state	The enactment state that consists of: • An enactment flow pointer. • The local data state. • The process condition state reflecting the satisfaction of process constraints. The process state is changed through enactment of a process step, interaction with a cooperating process, or initiation, suspension, resumption, or termination by a controlling process.
Project	An enactable or enacting process whose architecture has control processes (project management) and enacting processes performing the project tasks.
Project management	An enactable or enacting process whose goal is to create project plans and, when authorized, instantiate them, monitor them, and control their enactment. These responsibilities are commonly known as project planning (i.e., development of process plans) and project control (i.e., process evaluation of plan information and process control to make adjustments, if necessary).

Project manager	A human agent enacting the control process responsible for the execution of a project.
Redundancy	A process task or step that is not required by an error-free enactment. Redundancy thus compensates for human or other errors in process enactment.
Robustness	The degree to which the process rejects unauthorized process control and/or modification (intrusion).
Role	A subprocess definition. A process definition may consist of (descriptions of) subprocesses that interact or of subprocesses that create and control the enactment of other subprocesses. An agent, when assigned to performing the subprocess, is referred to as assuming the role. In this role, the agent is limited to the set of operations reflected in the steps of that subprocess (script for a human or program for a machine).
Task	A process (step), typically enacted by a human, requiring process planning and control.

REFERENCES

Feiler, Peter, and Watts Humphrey 1991 — *Software Process Definitions Draft Document.* Pittsburgh, Pennsylvania: Software Engineering Institute, Carnegie-Mellon University.

Guindon, R. 1988 — *A Framework for Building Software Development Environments: System Design as Ill-structured Problems and as an Opportunistic Process.* MCC Technical Report STP-298-88.

Parnas, D.L. 1972 — On the Criteria to be Used in Decomposing a System into Modules. *Communications of the ACM.* 15,12: 1053-1058.

Potts, C. 1989 — A Generic Model for Representing Design Methods. *Proc. 11ICSE.*

This page intentionally left blank.

BIBLIOGRAPHY

Britton, K.H., R. A. Parker, and D.L. Parnas. "A Procedure for Designing Abstract Interfaces for Device Interface Modules," *Proc. 5ICSE.* 195-204, 1981.

Clements, P.C., R.A. Parker, D.L. Parnas, J.E. Shore, and K.H. Britton. *A Standard Organization for Specifying Abstract Interfaces*, NRL Report 8815, June 14, 1984.

Dijkstra, E. W. "Co-operating Sequential Processes." *Programming Languages*, Edited by F. Genuys. New York: Academic Press, pp. 43-112.

Kellner, M. "Representation Formalisms for Software Process Modeling," *Proc. 4th International Software Process Workshop.*

Kirby, J. Jr., R. C. T. Lai, and D. M. Weiss. "A Formalization of a Design Process." *Proc. 1990 Pacific Northwest Software Quality Conference.* Oct. 29-31, 1990, 93-114.

Osterweil, L. "Software Processes Are Software Too." *Proc. 9ICSE.* March 1987.

Parnas, D.L. and P.C. Clements. "A Rational Design Process: How and Why to Fake It," *IEEE Transactions on Software Engineering*, February 1986.

Potts, C., and G. Bruns. "Recording the Reasons for Design Decisions," Proc. 10ICSE, April 1988.

Rendes, Barry, and Ralph M. Stair, Jr. *Quantitative Analysis for Management.* Third. Allyn and Bacon Inc., 1988.

This page intentionally left blank.